(2)

NEURAL NETS: A STUDY OF FACTORS AFFECTING THE ABILITY OF A BACK PROPAGATION NET TO RECOGNIZE ALPHABETIC CHARACTERS AND PRODUCE SPECIFIC SYMBOLS

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Clarence W. Potter, Sr.

~~November 1988~~
16 DEC 1988

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188
Exp Date Jun 30, 1986

1a REPORT SECURITY CLASSIFICATION
unclassified

1b RESTRICTIVE MARKINGS

2a SECURITY CLASSIFICATION AUTHORITY

2b DECLASSIFICATION / DOWNGRADING SCHEDULE

3 DISTRIBUTION / AVAILABILITY OF REPORT
Approved for public release;
Distribution unlimited.

4 PERFORMING ORGANIZATION REPORT NUMBER(S)

5 MONITORING ORGANIZATION REPORT NUMBER(S)

6a NAME OF PERFORMING ORGANIZATION

6b OFFICE SYMBOL (If applicable)

7a NAME OF MONITORING ORGANIZATION
HQDA, MILPERCEN (DAPC-OPE-A)

6c ADDRESS (City, State, and ZIP Code)

7b ADDRESS (City, State, and ZIP Code)
200 Stoveall Street
Alexandria, VA 22332

8a NAME OF FUNDING / SPONSORING ORGANIZATION

8b OFFICE SYMBOL (If applicable)

9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c ADDRESS (City, State, and ZIP Code)

10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
Neural Nets: A study of factors affecting the ability of a back propagation net to recognize alphabetic characters and produce specific symbols

12 PERSONAL AUTHOR(S)
CPT Clarence W. Potter, Sr.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Thesis	FROM _____ TO _____	88 Dec 16	86

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Neural nets, character recognition, back propagation

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

Neural nets can provide a user with the ability to train a system to accomplish a task. The net learns from the training pairs, and stores these relationships. The net provides a method to generalize an output based on the nearness to all of the trained inputs. This works well even in where the inputs are noisy or distorted. This is a study of a back propagation net. Neural nets degrade gracefully as the input deteriorates or gets farther and farther from the trained input. Hidden layers allow the net to store more complicated relationships. These relationships may well be something other than what the user thinks is the best or most obvious. This relationship is stored as connection strengths. The net applies an activation function to the weighted inputs at each node and then passes that information to the next node. This study shows how well the back propagation net accomplished the task of recognizing alphabetic characters.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS

21 ABSTRACT SECURITY CLASSIFICATION

22a NAME OF RESPONSIBLE INDIVIDUAL
Clarence W. Potter, Sr. CPT

22b TELEPHONE (Include Area Code)
(804) 734-3260 a/v 687-3260

22c OFFICE SYMBOL
ATCL-SAA

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

Neural Nets: A Study of Factors Affecting the Ability of a Back Propagation Net to Recognize Alphabetic Characters and Produce Specific Symbols

Clarence W. Potter, Sr., CPT
HQDA, MILPERCEN (DAPC-APA-E)
200 Stovall Street
Alexandria, VA 22332

Thesis, 16 DEC 88

A thesis submitted to the University of Tennessee, Knoxville, Tennessee, in partial fulfillemnt of the requirements for the Master's degree in Computer Science.

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature Clarence W Potter, Jr

Date 16 Dec 88

NEURAL NETS: A STUDY OF FACTORS AFFECTING THE ABILITY OF A BACK PROPAGATION NET TO RECOGNIZE ALPHABETIC CHARACTERS AND PRODUCE SPECIFIC SYMBOLS

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Clarence W. Potter, Sr.
November 1988

To Valerie
and Will

Abstract

Neural nets can provide a user with the ability to train a system to accomplish a task. The net learns from the training pairs, and stores these relationships. The net provides a method to generalize an output based on the nearness to all of the trained inputs. This works well even in where the inputs are noisy or distorted. This is a study of a back propagation net. Neural nets degrade gracefully as the input deterioates or gets farther and farther from the trained input. Hidden layers allow the net to store more complicated relationships. These relationships may well be something other than what the user thinks is the best or most obvious. This relationship is stored as connection strengths. The net applies an activation function to the weighted inputs at each node and then passes that information to the next node. This study shows how well the back propagation net accomplished the tasks of recognizing alphabetic characters.

Acknowledgements

I thank God for giving me the ambition and abilities to accomplish something of this magnitude. This has been a most interesting project, from which I have learned much.

The author is also very appreciative of the faculty of the Computer Science Department. Their assistance over the last two years has prepared me for this project. Dr. Bruce MacLennan, as my major professor, has been instrumental in guiding me through this endeavor. His expertise was essential in bringing this to fruition. Others on my committee, Dr. David Mutchler and Dr. David Straight, assisted me with guidance and plenty of questions to keep me thinking.

The love and support of my wife, Valerie, were essential to the completion of this thesis. She gave so much time and encouragement, and was often a sounding board for spur of the moment ideas that would pop into my head. The prayers of friends and family meant so much to this author. They all helped to keep this from being a droll process, and to keep it in perspective.

Thanks to Dorsey Bottoms, who saved me from the copier, in a time of desperation, just prior to my defense. I, also, appreciate Eric Kirsch for taking time to read and critique my work, and to help make my thoughts into something readable.

The gradlab crew who answered all of my endless questions about Sun work stations. They made it possible for me to get all of this done in a *reasonable* amount of time. Without them, this could have taken years.

Contents

List of Figures

Chapter 1

Introduction

The purpose of this paper is to describe the research and findings of a study on neural nets. The research was undertaken to learn more about a personal interest. The goals of this research were a deeper understanding of neural nets, suggestions to improve the nets that were tested, and a foundation for future research. As a topic, I found neural nets to be extremely interesting. These nets, although not a new concept, are, for the most part, not thoroughly understood. They have the potential to open a new look at computing and the way problems are solved. This study looked into a group of neural nets developed by Rumelhart and McClelland, as part of their Parallel Distributed Processing series.

This study will determine the effects of adjusting basic parameters and characteristics. In the forefront of our consideration are training, speed, and accuracy. Neural nets have to be trained. How much training does it take to get to a reasonable proficiency? What should we accept as a reasonable level of performance? How do you make the tradeoffs? What things should be traded and what should not be traded. The testing was designed to let the nets decide what

connections were important. Is it possible to tune these connections afterward to improve speed or performance? Neural nets degrade gracefully, but how robust are they?

Computers are getting faster and faster. This will allow for more nodes in the same amount of training or processing time. The simulations were designed to be simple and flexible at the cost of efficiency. It is still important to look at speed as it relates to the effects of changes, and how one net runs against another. Is bigger really better in a neural net? We will look at all of these questions in the chapters that follow. There will also be a general discussion of neural nets.

This paper is organized into four parts. It will provide some background information, explain the methodology for aquiring the data, discuss the significance of the results and problems, offer possible improvements in the implementation, and suggest areas for follow on work to continue this study. Chapter 2 is a general overview of net characteristics and strategy. Chapter 3 defines the specific type of net used, as well as the particulars of each net that was tested. Also discussed are the procedures used and justification of assumptions as appropriate. In Chapter 4 are the results of all of the testing and the problems that arose in the midst of testing. Finally, there is a summary and follow-on studies that I think will further help expand the potential of neural nets in Chapter 5.

Chapter 2

Why a Neural Net?

Why is there even an interest in neural nets? McCulloch and Pitts first discussed this idea in the 1940's. Minsky built a hardwired net that learned back in the 50's [14]. Nets were alive and well in the 60's. However, they were found to not be more than a novel idea for simpler problems, and interest in them waned. At that time, there were a lot things they would not do. It was not thought at that time that neural nets would open any new doors. Why then this resurgence in a dead issue? We should be headed to new frontiers in research. Still there is this question of how can we get computers to do certain mental tasks as well as a five-year-old [4]. Tasks such as pattern recognition, speech recognition, information retrieval, and visual perception [18] are done daily by children in seconds, but may take a computer much, much longer – if the computer can do it at all. Some tasks even stump the supercomputer [9].

Neural nets can provide an important tool in finding general solutions or in

finding solutions with noisy, distorted or incomplete inputs [11]. The net derives a relationship mathematically. The relationship does not have to be one that the user knows to exist. The function of the net is to find a relationship. The regularity detector is only given inputs. It then has to determine the similar parts, and whether the input is similar to the trained patterns.

A neural net is not given step-by-step instructions on how to solve a problem like other systems. It goes out and finds a solution. The net itself determines what is important. The system may even overlook relationships that are known to exist and find new ones. The knowledge is stored in the collective weights of the system, and those weights are not unique [21]. The system will probably find a slightly different solution each time it is trained. In life, different people see relationships in a set of objects from different perspectives, likewise neural nets can find different relationships. A neural net can use this to find outputs even in the face of noisy or distorted inputs.

How would a computer determine if a vector of 128 bits represents one letter out of a set of 26. With neural nets there may be 20-30% of the bits that are not like the computer expects them. How long would it take to make this determination? What would the output look like? This is especially relevant when the form of the output is directly related to the closeness of the input to the learned patterns. The pattern in question may be equally close to several other patterns. The neural net simply multiplies and sums its way from the inputs to the output. This task may require enormous time for large scale problem on a normal system, but will be performed quite directly with a neural net.

The idea for neural nets is to try to work in ways similar of the brain [20]. The brain is composed of very simple cells, passing simple messages to a lot of other cells, that collectively accomplish miracles. The human brain, with 10^{11}

neurons, can only accomplish about 100 steps in one second. The step refers, loosely, to computer operation type steps. At this slow speed it does things that computers have yet to do [17].

Neural nets have a lot to offer. One of the most important, is the ability of the net to generalize an output based on its training. It will find the similarities between the tested pattern and all of the trained patterns. The net can fill in data such as demographic information or some physical characteristics based on the other data entered.

In the case of pattern matching it can create an output for something it was not trained on. In one study that encoded family relationships, it actually internally represented the characteristics 'old' and 'Italian' that were never even mentioned [4]. The net finds relationships that may not be obvious on the surface.

Graceful degradation is when something starts to go down in performance or accuracy, but does not lose it all at once. Given poor inputs, whether from noise or distortion, the resulting output will be close to the trained output. On a standard computer system, testing of a pattern would be a Boolean result. Either it is or it isn't the pattern. It might give a quantitative measure, but has little way to give a specific output, that is generalized over large number of trained patterns. However, the net adds the shades of grey that the comprise the real world. The net will also give some value to the fitness of the output to those of the training set.

2.1 Basic Form

A neural net is a series of nodes, connections, and weights, that when properly trained can solve problems inspite of noisy inputs or incomplete data. The knowledge is stored in a distributed fashion in the connections between the nodes in the form of weights. Adjustment of the weights in the local connection produces learning in the global sense for the net [4]. The net takes a set of inputs and calculates an appropriate output based on its training. Figure 2.1 shows the general flow of information. The net multiplies the inputs by a weight stored at each connection. This product is then summed over all of the other connections at that node.

$$net\ input_i = \sum_j (activation_j * weights_{ij}) \tag{2.1}$$

$$output_i = f_i(net\ input_{ij}) \tag{2.2}$$

Equation 2.1 shows the simplest form. That equation does not account for the activation function. Equation 2.2 shows how the activation function comes into play. For each node j that sends information to i, there is an associated weight. The inputs to i from all j are multiplied by the weight on the connection from j to i These are then summed for all j, to give you the net input to i. The activation function is then applied to the net input. Activation functions are discussed in the following section. The process continues for all inputs and nodes down to the output level. The weights are generally derived from the training process, however they may be computed directly. The process to compute the weights may well be tedious for anything more than the simplest applications.

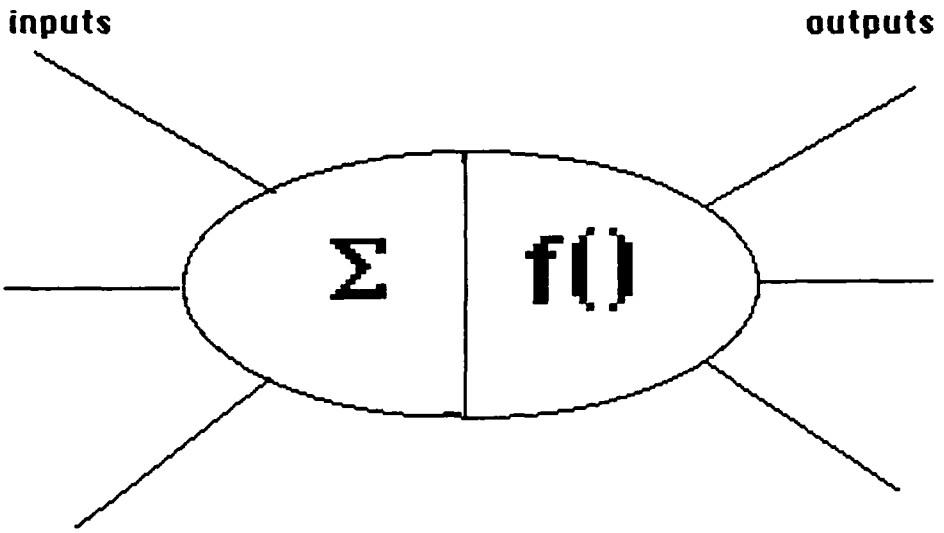


Figure 2.1: Basic data flow.

There are a number of different mechanisms on which the net can be operated. The implementation may be found in different forms. One company has built the net from hardware [7]. This net is made from VLSI chips on a card that plugs into an expansion slot on a personal computer. It can recognize hand written numbers. Most of the time software simulations are used that can be modified more readily than hardware. Looking to the future, we can expect to see neural nets in optical technology[1], molecular computing, and field computers [13]. Whatever the medium, the process is rather straight forward – that of summing weighted inputs through the net to produce an output.

Inputs to a net might represent any number of things, such as image patterns, sensor data (heat, seismic, pressure, etc), radar signals, weather data, and

audio/video signals. The outputs generally range from -1 to +1. Large numbers will often be represented by numbers that are scaled to fit the interval. Inputs and weights may be constrained to be positive, negative, zero, continuous, or discrete. For the nets used in this project, all inputs, outputs, and weights, are continuous between -1 and 1. Nets also may have a variety of connection strategies with which to pass information between levels and between nodes on a level.

2.2 Types of nets by activation functions

Nets are characterized by their different methods for determining there output. The output is a function of the sum of the weighted inputs. This function may be anything you wish, but is normally of the types linear, linear threshold, or logistic (sigmoid).

The linear function (Fig. 2.2) is a straight line function applying some constant to the sum of the net inputs to a node . The activation of a unit is directly proportional to the net inputs. Given an activation, the node multiplies by a slope, add an intercept, and send its output on to the next node. Linear functions are not used much because any layers of hidden units may be rolled back into one layer. Therefore the power of hidden units is lost. The power of the net to internalize a representation is reduced.

Another type of activation function is the linear threshold function (Fig. 2.3). These functions are constant up to some threshhold at which time they act in a linear fashion. These functions help to make up for the short fall of strictly linear nets. The hidden units give the ability to store some representation of the data. Without hidden units, it would be basically all input and output.

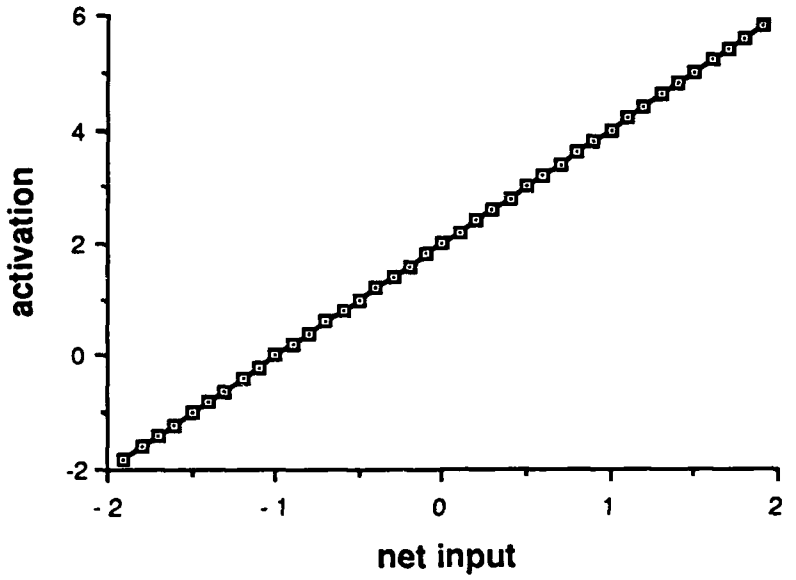


Figure 2.2: Linear activation function.

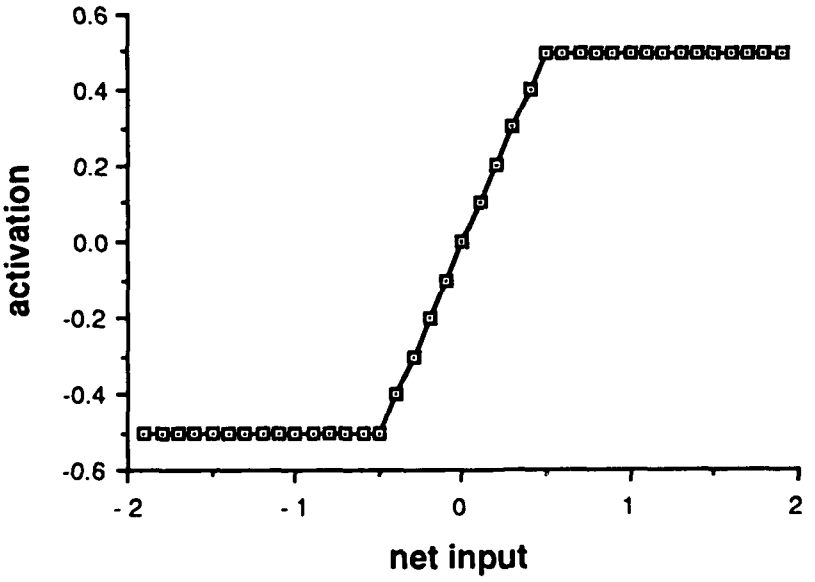


Figure 2.3: Linear threshhold function.

The logistic or sigmoid function is similar to the function on the right (above) except that it is a continuous function. The function is defined in equation 2.3.

$$output_j = \frac{1}{1 - e^{-net_j}} \tag{2.3}$$

The equation came about as a realization of the need for a continuous function similar to the linear threshhold. The ability to use derivatives of the activation function are important to the back propagation net. That net is used exclusively in this project.

2.3 Training the net

To train the net, inputs and outputs are loaded into their respective areas. The net then determines a difference between the calculated output and the target output. The basic form of learning adjusts the weights relative to the difference between the target and the actual output (Equ. 2.4, 2.5). The learning rate is a constant of proportionality. The rule for back propagation nets is different in that it includes the derivative of the activation function.

$$delta_i = learning\ rate * (target_i - activation_i) \tag{2.4}$$

$$delta_{ij} = f'_j(net_{ij} * \sum_k (delta_{ik} * weights_{kj})) \tag{2.5}$$

This difference may be propagated back through the net in a number of different ways. We will look briefly at the different types of training functions,

paying particular attention to back propagation. In the back propagation net, the error from the higher unit is carried down to the lower unit which assigns the change in its weights as a portion of the error from above.

The simplest form merely increments or decrements the weights by a predetermined amount, based on the difference between the calculated output and the target. If the output is greater than the target, then all of the weights are decremented by a specific amount. How the amount is determined, gives way to the different methods for training the net. The amount may be preset, or calculated by any number of methods. The Hebbian learning rule multiplies the activation by the weight, and then by a learning rate parameter (Equ. 2.6).

$$delta_i = activation_i * output_j * learning\ rate \tag{2.6}$$

2.4 Other characteristics

In addition to activation function, and training, there are other characteristics of nets such as feedback, connectivity, and number of levels to consider in designing your net. Each of these will be addressed in the following paragraphs.

In most nets the connections carry inputs from a lower unit to the next higher level. Feedback occurs when the higher level is allowed to send its output back to a lower level. This is most often seen in auto associator nets while completing the pattern. The feedback helps strengthen the partial pattern in the layer below. This way, at each pass through, the inputs are closer to the completed pattern.

This is also necessary when some level of semantic understanding is desired.

Very often understanding the context has a lot to do in determining the words and phonemes at a lower level. As words are beginning to be activated, the overall meaning is also being formed at a higher level. If a word supports the overall meaning it is reinforced with feedback, otherwise it is inhibited. Feedback provides the necessary intermediate information to assist the net in finding a solution.

Since the training process is computationally intensive for large nets, it is necessary to determine the net configuration that best defines the problem. The fewer connections the less processing time that is required for each learning or testing cycle. Making this determination can be a very difficult process. Part of this study includes whether increasing the number of levels improves performance. This will be discussed in the section on results. Increasing the number of connections, or the levels will increase the complexity of the net and the time required to cycle through the training. Increasing the number of nodes increasea the floating point multiplications in proportion to the number of connections each node has. A fully connected net that increases the number of nodes by n will increase computation by an order of n^2. It may, however, be beneficial and necessary in the final running of the net to include all of the connections. This allows the net the most flexibility in determining the internal representation.

2.5 Current/future applications

This section addresses general ideas of current research. It is only intended to give a flavor for the kinds of studies being done. Pattern recognition is probably the most common use of neural nets. Nets can learn different types of patterns. By

patterns, you could mean a pattern of audio/video signals or other types of sensor input. It should be stressed that the nets can handle more than visual patterns. The net doesn't know the difference. For the net, a number is a number. For the time being we will discuss visual patterns of pixel images. Difficulties arise when trying to make the net recognize patterns that are not in the normal orientation or size. One such study has achieved good results in picking numbers when several may be present [6]. It cycles through picking the strongest first, and then following up with the rest. In that study, the patterns were scaled to different sizes and not placed in the center of the image area.

Another area of interest within the pattern arena is in edge detection. Often, it is hard to identify a pattern until you separate the pattern from background and other interferences. Once the edges are detected, those edges can be inputs to a net that identifies the object. Humans identify animals by their specific parts, such as a leg, foot, wings, etc. Once parts are identified, it is easier to see the rest of an object, and identify that object.

Speech understanding is another hot and very difficult topic. Abig problem is that no two people speak exactly the same. Even the same person does not always say the same words with the same tone and inflection. Having a net that can identify words is not sufficient, since many words have similar spellings and pronunciation, but different meanings. The net must in the end apply some contextual reasoning to the words it has identified. This goes back to the importance of feedback.

One project also takes written material and finds the phonemes that make up words. The net is good even to the point of words that are unusual or are pronounced differently than those with similar spellings. The phonemes are then used as input into a speech synthesizer [4].

The net's ability to determine unseen relationships would make it an excellent tool for economic forecasts. What would be better than a neural model that is designed to determine relationships and internalizing them? This would be especially interesting, since no one really knows exactly how the economic system works. The model may have difficulty predicting a catastrophic event, such as a crash. There may not be enough information for the net to learn all of the causal relationships. There is a net that runs financial data for a loan company to determine whether a person is a good loan risk [19]. The net was trained with 240,000 actual case histories of good and bad loans. The system was then tested with 30,000 other case histories. The system improved profits by 12%.

Neural nets have also made their way into robotics. There the net is used to control an arm that to grasp a cylinder in space [12]. The net was taught how to move the arm. The net then controls the three motors that move the arm. The arm was accurate to within 4°.

Chapter 3

Testing Procedure

3.1 Terms, abbreviations, and notation

input pattern – 128 bit vector that forms an alphabetic character when placed into a 16x8 matrix.

output pattern – vector of 128 floating point numbers between -1 and 1, that represent the solution of the net.

target pattern – 128 bit vector that forms the block symbol that the net responds with when the net recognizes a character.

pss – pattern sum of squares. The difference between the output of the net and the target summed up over the 128 locations for one pattern.

tss – total sum of the squares for all of the patterns in a group.

epoch - training cycle where all of the patterns in a group are entered into the net. The cycle includes calculating the output, and adjusting the weights.

level - number of 128-node connections that make up the net. Net 2, for example, has 2 levels of connections - one from input to the hidden units and another from the hidden units to the output units. For level 3 and 4 there is an additional designation of 'out'. Where the added level widens the middle level instead of making it deeper.

group - the number of pattern/target pairs used in training or testing. Patterns are grouped into 2, 4, 6, 8, 10, 12, or 14 patterns. Higher level addresses more nodes and deeper, where larger applies to group size.

input unit - units that receive the input or testing patterns from a source outside of the net.

output unit - units whose activations represent the solution of the net.

hidden unit - units that are neither input nor output. Units that are not directly reached by things external to the net.

notation - The following format explains from where certain outputs were obtained. The labels tell which level net, the maximum value of weights that were zeroed, and the percentage of distortion of the input pattern. They are seen throughout this paper without further amplification.

net level	zeroed weights	.	percentage distorted
2	15	.	20

zeroed weights that were $\leq = +/- .15$

percentage 20% of the bits of the input pattern were flipped

3.2 Net description

The implementation used for this project was the back propagation model developed by McClelland and Rumelhart [16] as part of a series of neural net models. The reference [16] provides source code in C, on $5\frac{1}{4}$" floppy disks (DOS formatted).

The net is fully connected from one level to the next. This allows the net the most flexibility in solving the solution. The net will determine which connections are needed and set the weights on those connections accordingly. In this way, the net internalizes the relationships. Connections that are not important are near zero. Those that negatively affect the outcome are given large negative values. Important connections are large positive values.

The tests were run on similar nets, but with different sizes and structures. Each level has 128 nodes (Fig. 3.1). The 'out' levels added nodes to the middle hidden layer. Actually, the 'out' levels are a level 2, but for testing purposes are the same size as the level 3 or 4 nets. All other increases add depth to the overall net by adding a new hidden level. Hidden units are those that give the net more power than a simple single level net. All of those neurons that are not inputs or outputs are the hidden units.

The patterns form a 16x8 matrix as pixels for an alphabetic character. The patterns are created by a series of 0's and 1's to fill in the shape of a printed letter. The simulation uses '**' to represent 1's for output. In the charts, each location is 2 characters long. The number, whether one or two digits, represents the output at that location multiplied by 100. All output values are within the range [-1,1].

level	configuration					total
	input	hidden			output	nodes
1	128				128	256
2	128	128			128	384
3	128	128	128		128	512
3out	128	256			128	512
4	128	128	128	128	128	640
4out	128	384			128	640

Figure 3.1: Number of nodes in levels tested.

Figure 3.2 gives some examples of input patterns. The complete set can found in the appendix. The last pattern is checkered to give the system a different type pattern to deal with, and to demonstrate the flexibility of the neural net. There is also a blank pattern.

The targets are six 8x8 block patterns (Figure 3.3). These six blocks are oriented in the upper or lower region of the 16x8 pattern. In addition to these 12, there is a blank target and checkered pattern, for a total of 14 targets. The blocks are designed to be sufficiently different that they may be identified, even when moderately deformed. The targets also are intended to make full use of the entire target space area, and not overly emphasize any particular area of the pattern space. As in the input patterns, one pattern was blank, and another checkered to see how the net would react to an unusual output. The checkered output pattern is the reverse of the checkered input pattern. Different size groups were used to see how the nets reacted over a range of difficulty. The groups range in size from 2 to 14 patterns.

The next figure (Fig. 3.4) shows the density of individual elements in the target space. From this picture we would hope to get some idea as to any unusual

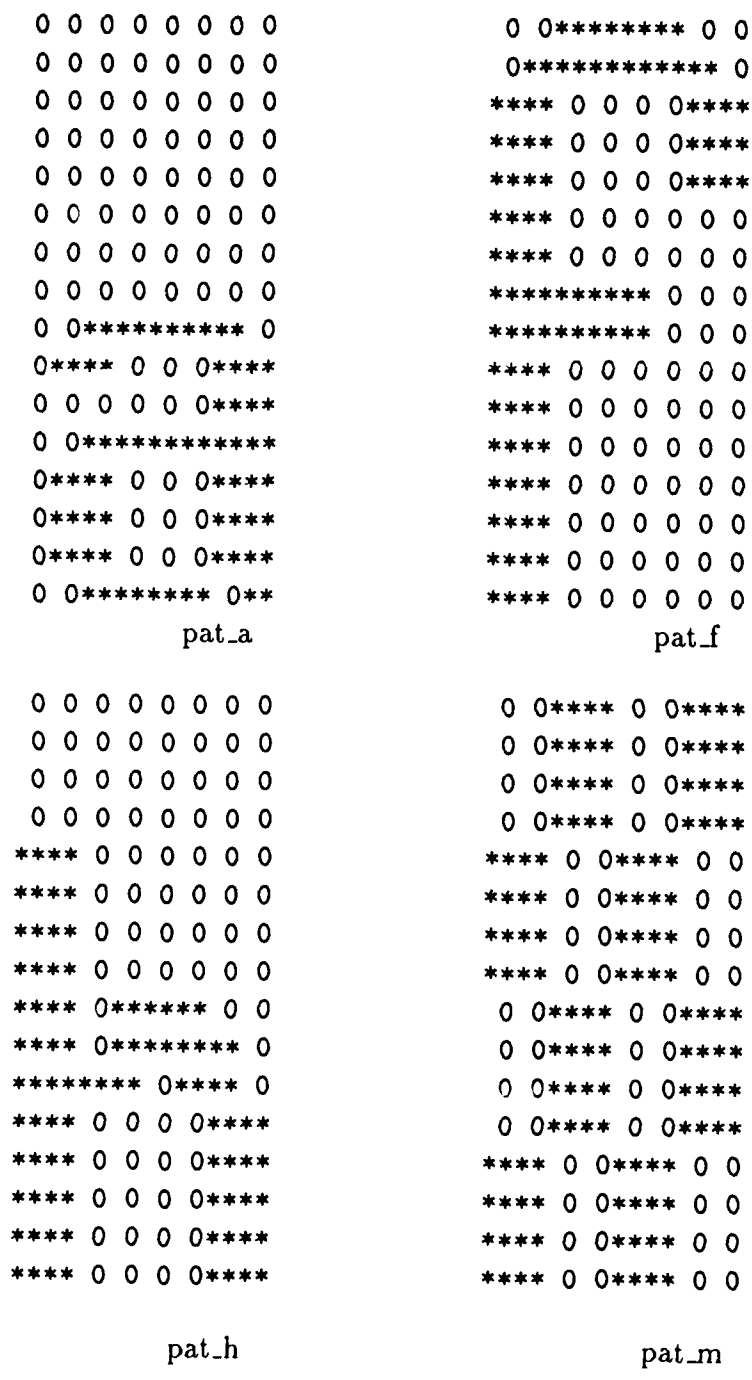


Figure 3.2: Examples of input patterns.

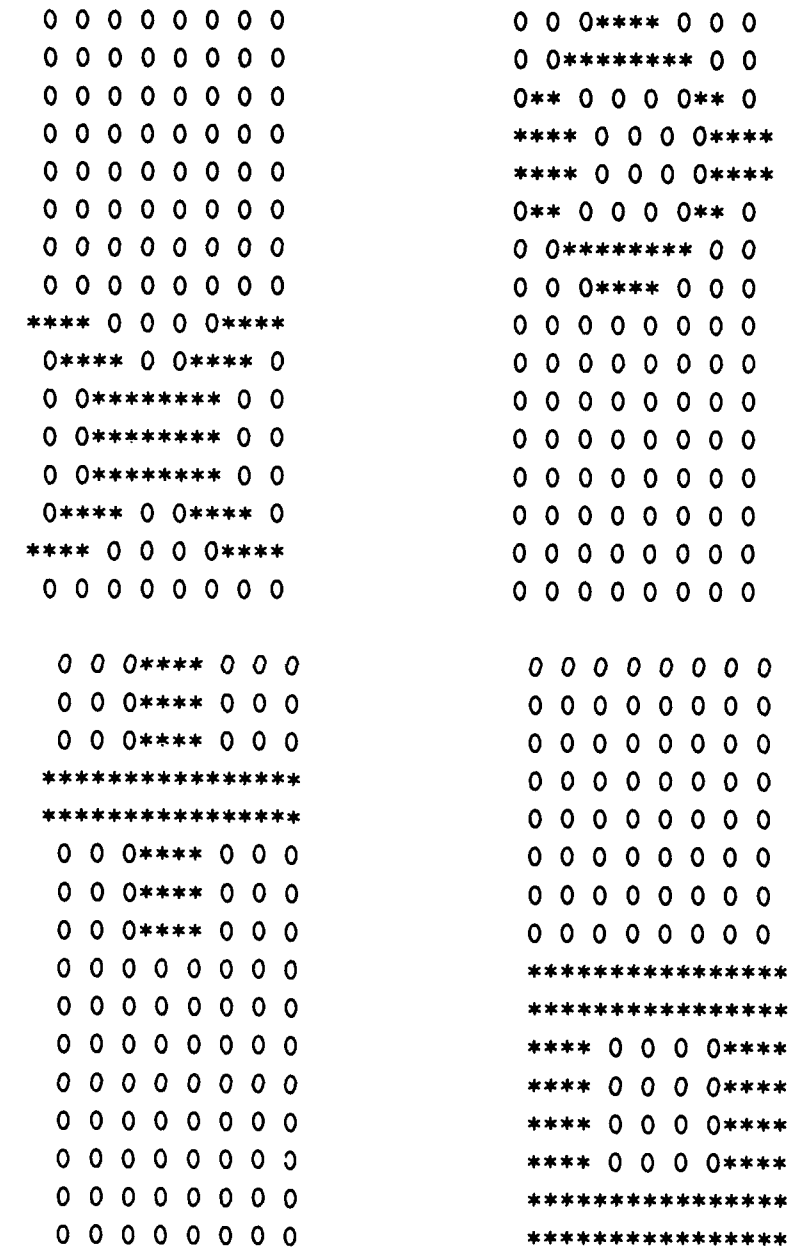


Figure 3.3: Examples of the target patterns

```
3 3 1 4 5 2 2 2
2 3 4 4 5 5 2 1
2 3 3 4 5 4 2 1
4 5 4 3 4 5 4 3
3 4 5 4 3 4 5 4
2 4 3 3 2 2 5 3
3 3 4 5 4 3 4 4
2 2 3 5 4 2 3 3
3 3 1 4 5 2 2 2
2 3 4 4 5 5 2 1
2 3 3 4 5 4 2 1
4 5 4 3 4 5 4 3
3 4 5 4 3 4 5 4
2 4 3 3 2 2 5 3
3 3 4 5 4 3 4 4
2 2 3 5 4 2 3 3
```

Figure 3.4: Solution density of target space

net characteristics that may be due to the distribution of the patterns. It appears that the numbers are not clustered in any particular area.

3.3 Back propagation

Linear activation functions with multiple levels can be rolled back into one level. There is, therefore, no gain in using multiple hidden layers to try and discover deeper more complex relationship. For back propagation to work, the function must be continuous and the derivative must exist. With these requirements, linear threshhold functions are also excluded. The activation function for back propagation is the logistic activation function (Equ. 3.1, Fig. 3.5). This function allows the most change when in the midrange of its inputs [17]. The node is considered 'undecided' when in that range.

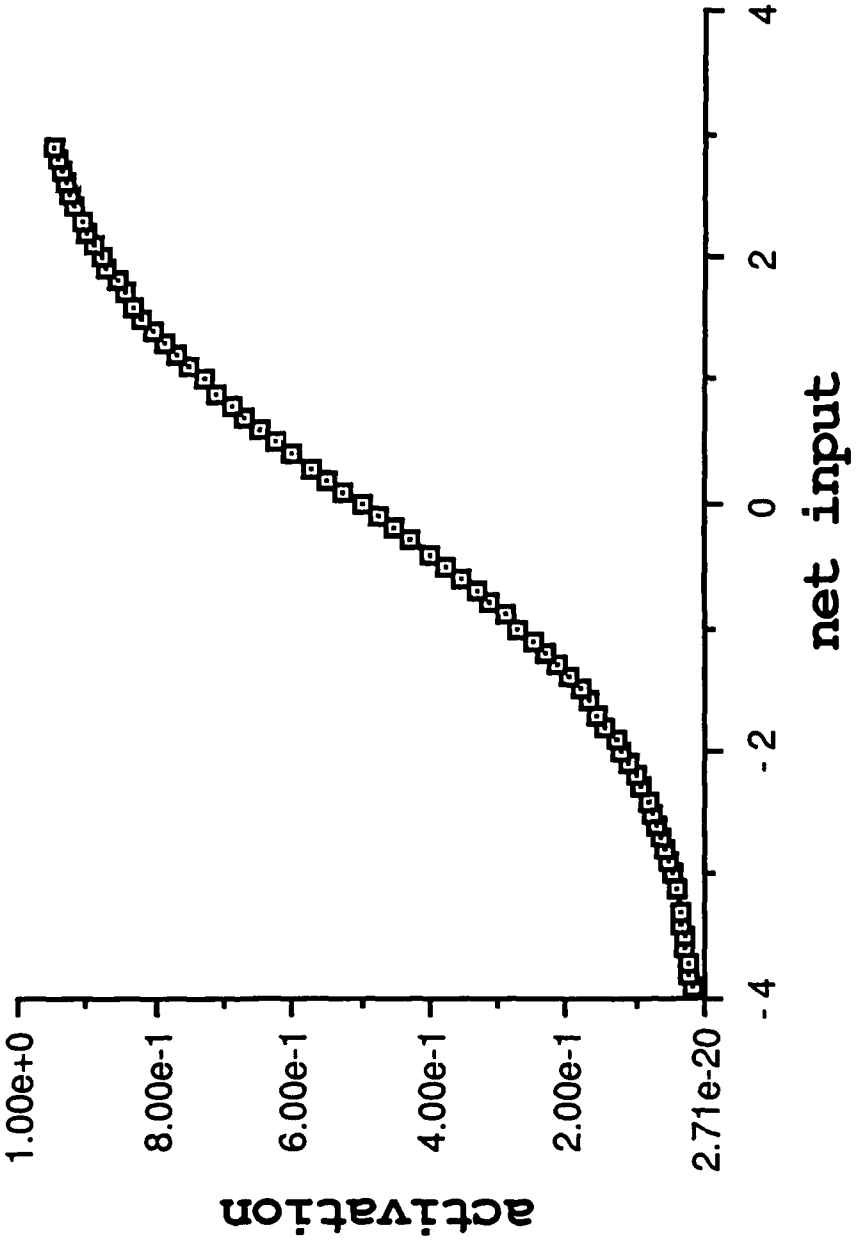


Figure 3.5: Logistic activation function for back propagation nets.

$$activation_i = \frac{1}{1 + e^{-net_i}} \tag{3.1}$$

BP is able to learn different types of functions as alluded to in Chapter 2. BP can learn signal filtering functions to be used with electrocardiograms (EKG) [21]. It has also been used with optimizing squashing functions for passing data through narrow bandwidth channels [4].

Back propagation (BP) is a gradient descent technique. Therefore, it will seek a local minimum. We will look to see if BP gets trapped in a local minimum. Ideally, the net should always be reducing the error level for the net.

The system works in two passes. The first pass is a forward pass from the inputs to the outputs. During this pass an output is calculated. The second pass moves from the output back to the inputs. The derivative of the activation function is used to determine the delta value from the difference between output and target at each node.

Back propagation learns by passing back an error value which when multiplied by the input from a previous node determines the change in that weight for that input. Equation 3.2 is the delta rule for use with output units in a back propagation net. In it you see that we are solving for the weight from j to i. It is the j portion of the error value that gets changed. The equation is applied for each node with a connection to the current (i) node. The delta function for hidden units without an explicit target is shown in Equation 3.3.

$$delta_{ij} = f_j'(net_i) * (target_i - activation_i) \tag{3.2}$$

$$delta_{pi} = f_i'(net_{pi} * (\sum_k delta_{pk} * weight_{ki}) \quad (3.3)$$

3.4 Testing

Training levels

The patterns were tested in groups at each level to determine how much training was required and what error levels could be achieved. Since bigger groups and higher levels require much more processing time per cycle than smaller lower ones, it is important to know if additional training will significantly improve the performance. Each pattern group was run for 500 training epochs. Error levels were checked after each epoch, for the first 10, after each 5 for the first 50, and after each 100 for the first five hundred. Selected groups were also run with different random seeds to determine the effect of different initial random weights. The levels were tested on the each of the 6 nets (1, 2, 3, 3out, 4, and 4out). The final weights were saved for the testing phase.

Pattern testing

After training, the nets were subjected to a variety of tests. The first was to test the net with distorted patterns. The strong point of neural nets is to identify the patterns in spite of noisy, incomplete, or distorted inputs. Incomplete patterns were considered to be a subset of the distorted patterns. The patterns were distorted by flipping random bits from one to zero, and vice versa. The same mask was used for all patterns. Masks changed 10, 15, and 20% of the bits. Each of the distorted pattern sets were tested against all 7 group weights, on each of the six nets.

Weight reduction

Since the size of the nets increases the amount of floating point multiplications, the time required to evaluate the inputs will also be increased. To reduce the floating point computations, the weights were set to zero if they were close to zero. Weights were zeroed that were less than or equal to +/- .05, .1, .15, and .2. The adjusted weights were then retested against clear and distorted patterns.

Speed

By their very nature, nets are large and have many connections. All of these are floating point values. The nets are cpu time intensive. For this reason, time tests were run on each size group, on each net to determine the average time to process one pattern. The system did not allow for an accurate measure of individual cycles, so 100 epochs were used to average out the inaccuracy of the timing. Similar tests were run on several Sun work stations to include: Sun 3/280, Sun 3/60, and Sun 4/110. Time tests were also run on a Vax 8650, and a Vax 8200 both of which run Ultrix (2.4 FT1 on the 8650 and 2.2 on the 8200).

Chapter 4

Results

Rumelhart and McClelland did a good job of building almost everything you could hope for. After a little testing, you find you have more data than ever expected. This section will layout the significant discoveries of this research. The data and graphs are representative of the data and are not intended to show every case. The examples are not the only time when a specific incident occurred. There is also a section that will address the problems in greater detail.

4.1 Training curve

There were several interesting things noted about the learning curves. The first (Fig. 4.1) was a rather uniform and parallel progression of the tss from small group size to large. The more different patterns you have the harder it is for the net to get them all correct. This was not always the case. On the first run of the level two net, several larger groups did better than smaller ones.

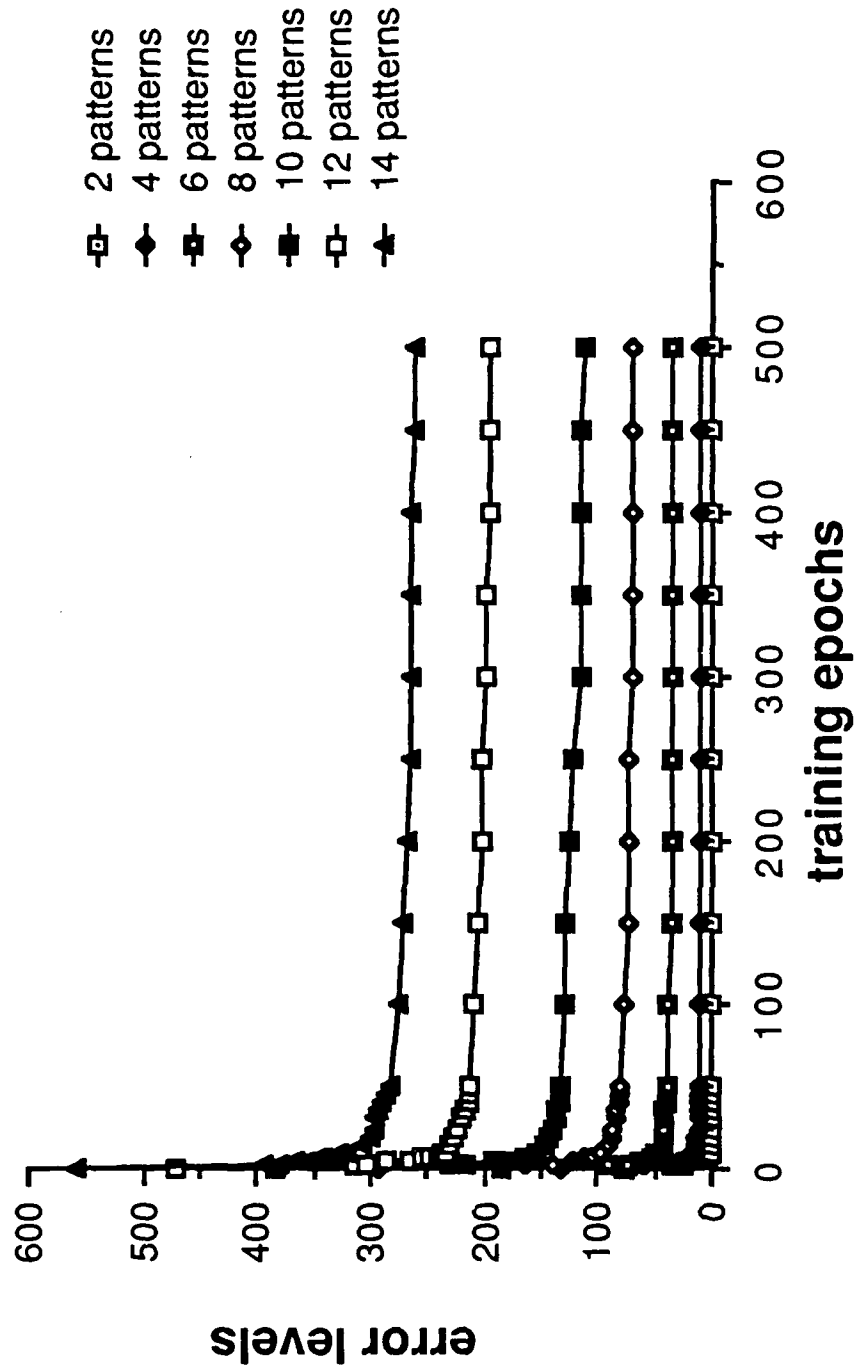


Figure 4.1: Level 1 training curves, all size groups.

Specifically, the groups of size 8 and 12 were better than those of sizes 10, 6, and 4 (Fig. 4.2). The higher level nets (3, 3out, 4, and 4out) performed terribly. The error never improved significantly. It stayed near the initial level or slightly increased throughout the entire test. Further study revealed that it did make one good drop after the first training epoch, then it oscillated about a relatively straight line.

Two runs of the 10 pattern group in a level 2 net, did not show similar training curves. This caused some doubt in the validity of the runs. To satisfy this, multiple tests were run on each size group to insure that the results were in a normal range. Of 5 tests run, each with a different seed, 4 of the 5 were in a tight group throughout the entire curve, while the fifth ended the run with an error more than 4 times the group average (Fig. 4.3). This indicates a serious problem in choosing a seed. Even though back propagation is a gradient descent technique, it can have real problems converging. All tests have been rerun to either show differences in error in similar runs with different seeds or to show general trends with a specific seed. This seed remained the same for the duration of the tests.

The nets did the majority of the learning in the first 10 epochs based on the slope of the learning curve. The slopes were nearly flat by 50 epochs, although there were several instances where there was a late drop (Fig. 4.4). This can greatly reduce the time to train and test the nets. There may be also be a reasonable cutoff that can be determined by some heuristic method. Little additional learning occurred after 100 epochs.

Another curious outcome was that even with a standard seed runs with more patterns would do better than a run with fewer patterns. This was seen on several occasions, sometimes the improvement was temporary and the smaller net did better in the end. Other times the larger net was better the entire time of

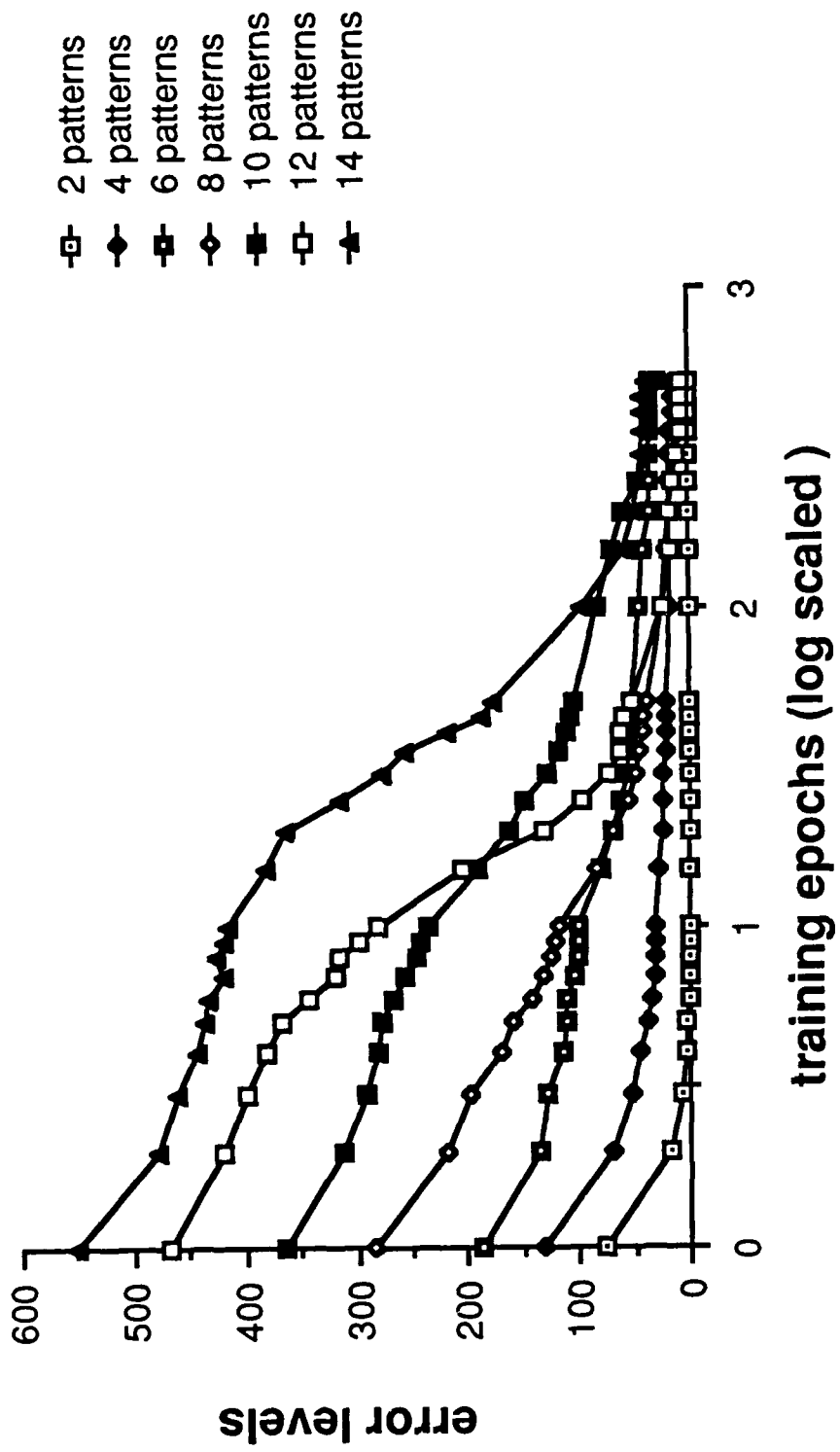


Figure 4.2: Level 2 training curves, all size groups.

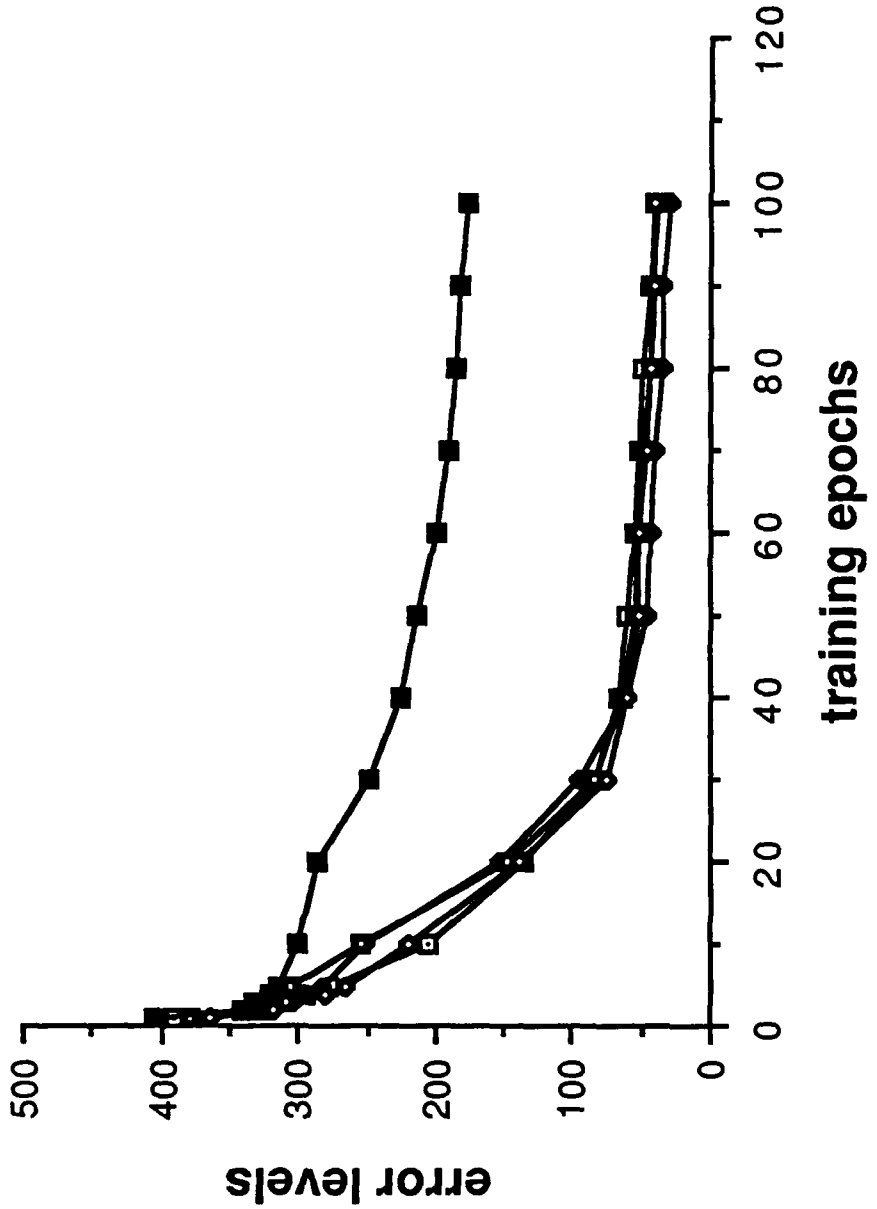


Figure 4.3: Graph of Level 2, 5 runs of 10 patterns.

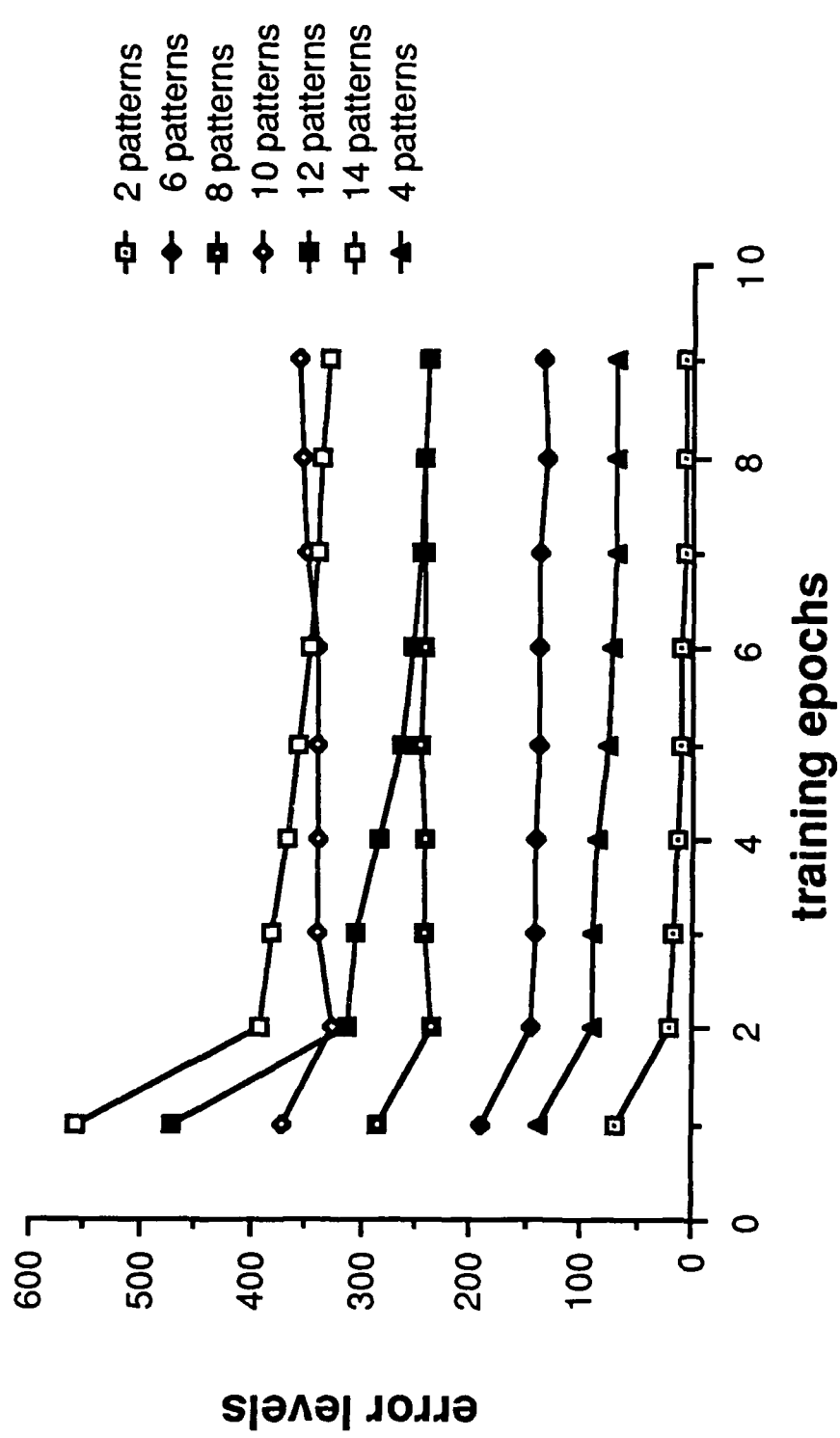


Figure 4.4: Level 3 training curves, all size groups.

testing. This may be attributable to some targets going against the grain of the net. It is possible that adding other patterns later may have helped the net to get over that hurdle. The later patterns in effect gave the net more information on which to internalize the relationships.

In summarizing the lessons learned concerning the amount of training, we find that during the initial stages of each run, larger size groups caused higher error rates. Each size group, generally, converged uniformly. However, in the later stages, there were instances where larger size groups performed better than smaller. Most of the training was completed by the 50th training epoch. Little additional improvement occured after the 100th epoch. The first 10 epochs gave you a good idea of how the rest of the graph would behave. The section on momentum provides some improvements and insights to the problems of the net's ability to reach a minimum.

4.2 Momentum

The most significant discovery was found in the higher level nets (levels 3 and 4). Initially the problem with the higher levels not improving was attributed to 'simpler is better' - major discovery. That was not likely, though. Higher level nets can duplicate lower nets. A higher net can do the same as the level 2 in the second level, and then just pipe the solution through. So, higher level nets should be better, up to some limit where the time required to train so overwhelms the gain that it is no longer cost effective. It was also a concern that there was just too much to sort through to always find optimal solutions. There are so many weights. If they are all allowed to change a little, it is no wonder that they might

cancel out the progress made by other changes and cause oscillation.

The higher level nets did not perform well, as far as tss goes. There was one good drop on the second epoch and then the error rate was headed back up. Throughout the training the higher nets in the larger groups would oscillate. They did not produce any kind of trend that would reduce the error. And, in fact, the error curves were slightly increasing (14 patterns curve, Fig. 4.5). Thus arises one drawback to neural nets: they can get caught traversing error ridges and never get headed down into the reasonable solution space. This leads to a future topic of heuristic methods to determine seeds/biases, weight changing and training strategies. It might, also, be better to train on one pattern at a time, and then train the next, or in some way over-emphasize patterns that are harder to learn.

More reading found the almost unnoticed parameter momentum. This low visibility variable, momentum, gives the $delta_i$ the actual amount of the change to the $weight_{ji}$ (Equ. 4.1). Tests showed vast improvements on the 14-pattern groups, when the values for momentum went from the default of .9 to .6 and .3 (Fig. 4.6, 4.7, and 4.8). Even these values were not sufficient for nets of level 6 or 7. At $momentum = .3$, a level 7 net oscillated severely (Fig. 4.9).

$$delta_{ji} = momentum * f'_j(net_i * weight_{ji}) \tag{4.1}$$

4.3 Discrepancy in the tss

The following shows relative distortion of the output patterns. The right sides are the targets. The lefts are the outputs. The tss shows the relative error.

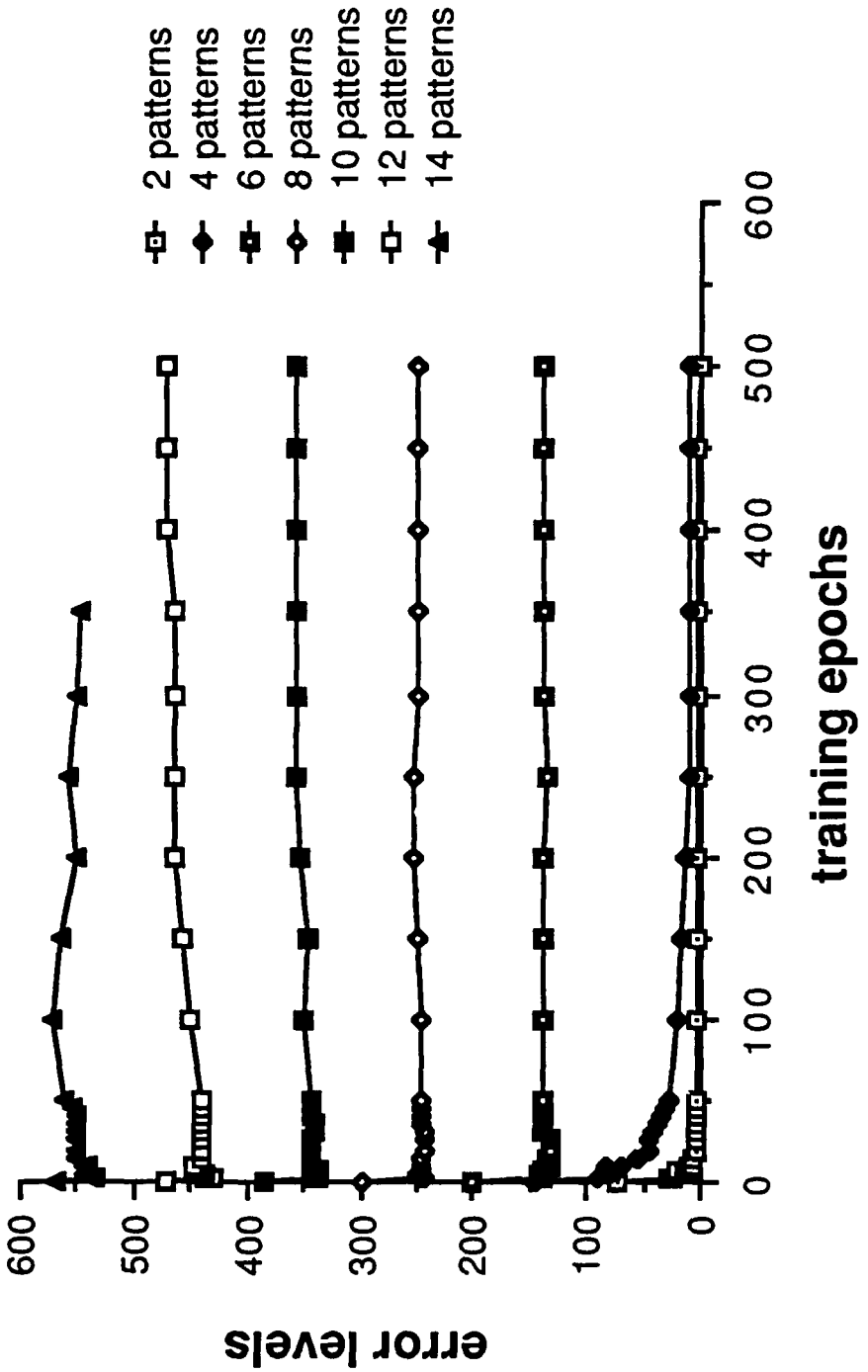


Figure 4.5: Level 4 training curves, all size groups.

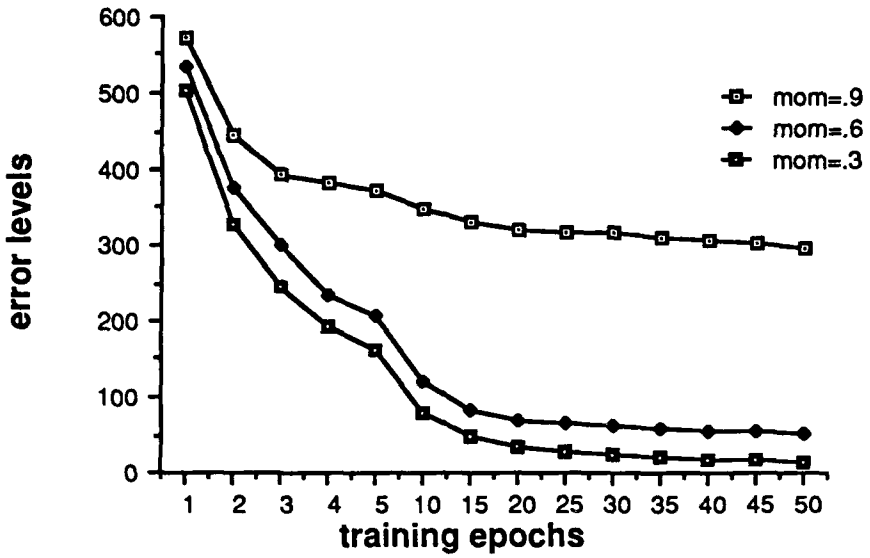


Figure 4.6: Varying momentum in level 1.

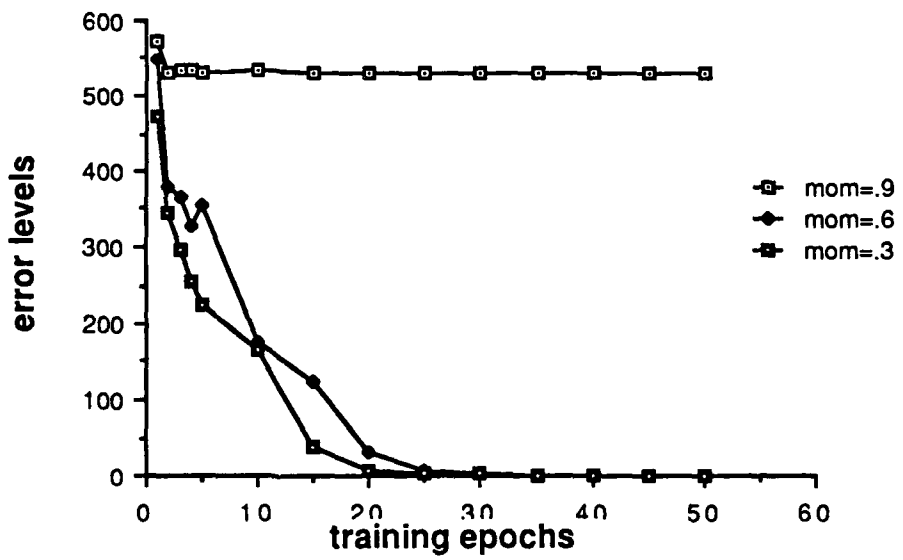


Figure 4.7: Varying momentum in level 3.

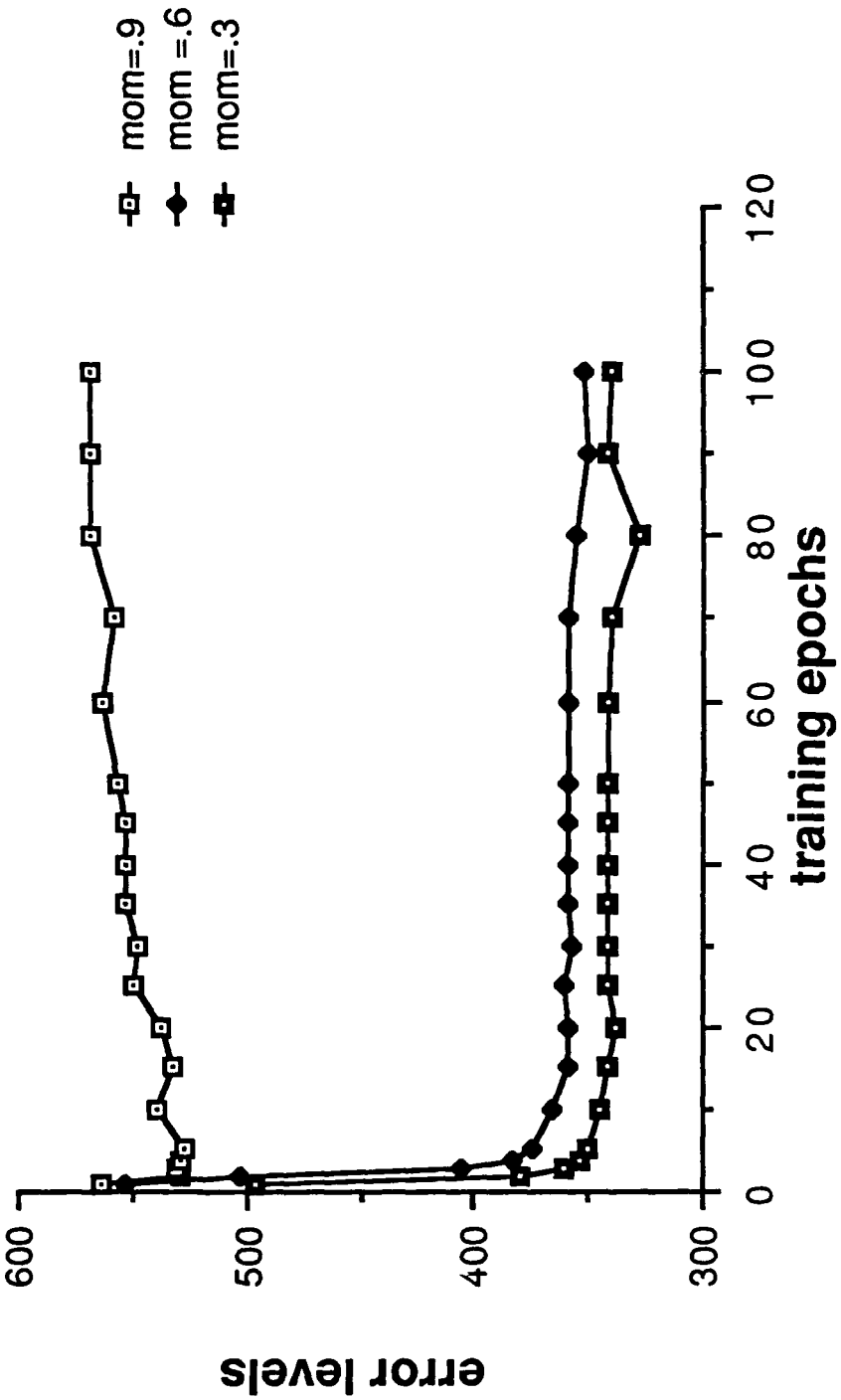


Figure 4.8: Varying momentum in level 6.

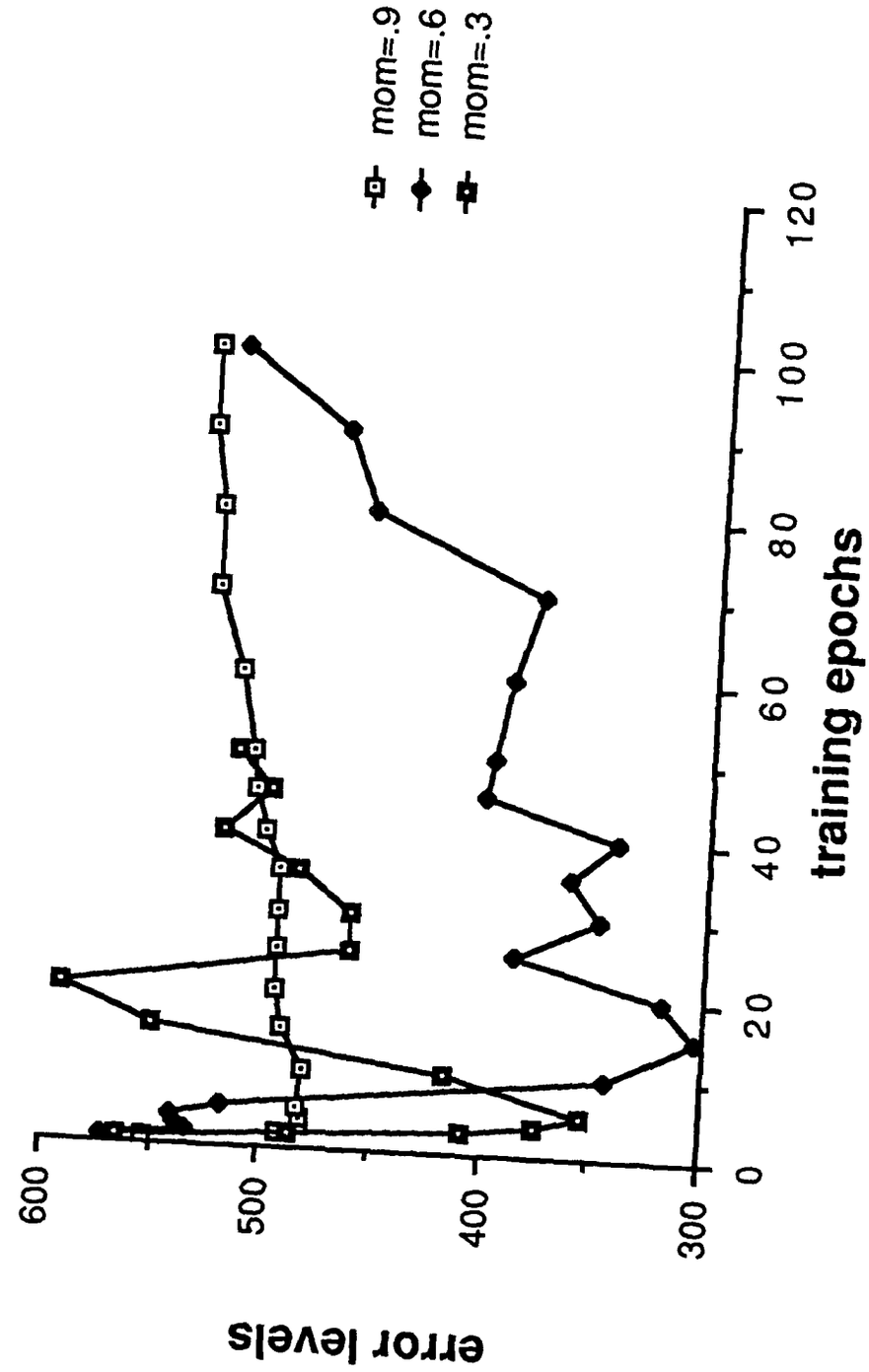


Figure 4.9: Varying momentum in level 7.

This error is not always a good measure. The first and second, output_i and output_m (Fig. 4.10), show close errors, but the first is really not distinguishable. In the second, you can see the resemblance. Notice that although output_b (Fig. 4.11) and output_l (Fig. 4.12) have very large differences in the tss, they don't appear to be that different from their respective targets. Output_l (Fig. 4.12) and output_e (Fig. 4.13) also have close error levels, but the former is not near as distinguishable as the latter.

There appears to be a connection between the high coverage area of pattern m and the high relative tss. This is based on the fact that with low coverage areas it is more likely that the 0's from the target will match those of the output.

There appears to be a sensitivity of the net toward the area shown in output_l (Fig. 4.10). Several outputs were distorted toward this general pattern. The same sensitivity did not exist in the upper block area. The density of the solution target space does not show a reason for the problem. [See the solution density pattern in chap 3 (Fig. 3.4).] These patterns have been filtered to remove some of the values where low activations registered to make them more clear. The blank spaces show the locations that were filtered.

4.4 Changing the percentage of distortion

During this test, patterns were distorted by changing a percentage of the bits from 0 to 1 and from 1 to 0. Each 5% changed 6 bits of the pattern. The same bits were changed for each pattern. Selected patterns are shown to demonstrate two things. First, that the change of a few bits did generally show a gradual decay in the performance, as expected (Fig. 4.14). Second, there was also a severe change at times when the few bits were changed (Fig. 4.15). Also note, at times

```
tss 12.697
8 0 0 0 0 0 0 0          0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0          0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0          0 0 0 0 0 C 0 0
0 0 0 0 1 0 0 0          0 0 0 0 0 0 0 0
0 0 0 4 0 0 0 0          0 0 0 0 0 0 0 0
0 065 0 0 0 0 0          0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0          0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0          0 0 0 0 0 0 0 0
0 04683 09680 0          **** 0 0 0 0****
0 0 03699361499           0**** 0 0**** 0
0 0 01099 09999           0 0******** 0 0
0 0 7 099489799           0 0******** 0 0
0 0 2779988 099           0 0******** 0 0
5 2 79996 0 057           0**** 0 0**** 0
927999762994269          **** 0 0 0 0****
999 3 1 92197 1           0 0 0 0 0 0 0 0

     output_il                  target_i

tss 14.9065
99 0 0 0 099 0 0         **** 0 0**** 0 0
 099 0 0 099 0 0         **** 0 0**** 0 0
 099 1 09999 0 0         **** 0 0**** 0 0
 0 0 0 099 0 0 0         **** 0 0**** 0 0
 0 09999 0 099 0          0 0**** 0 0****
 0 09999 0 09999          0 0**** 0 0****
 0 099 0 0 09999          0 0**** 0 0****
 0 0 0 0 0 0 0 0          0 0**** 0 0****
 0 09999 0 09999         **** 0 0**** 0 0
 0 09999 0 09999         **** 0 0**** 0 0
 0 09999 0 09999         **** 0 0**** 0 0
 0 099 4 0 09999         **** 0 0**** 0 0
 0 0 0 09999 0 0          0 0**** 0 0****
9999 0 09999 0 0          0 0**** 0 0****
9999 0 09999 0 0          0 0**** 0 0****
9999 0 09999 0 0          0 0**** 0 0****

     output_m                   target_m
```

Figure 4.10: Output from level 200.00, patterns i and m (filtered).

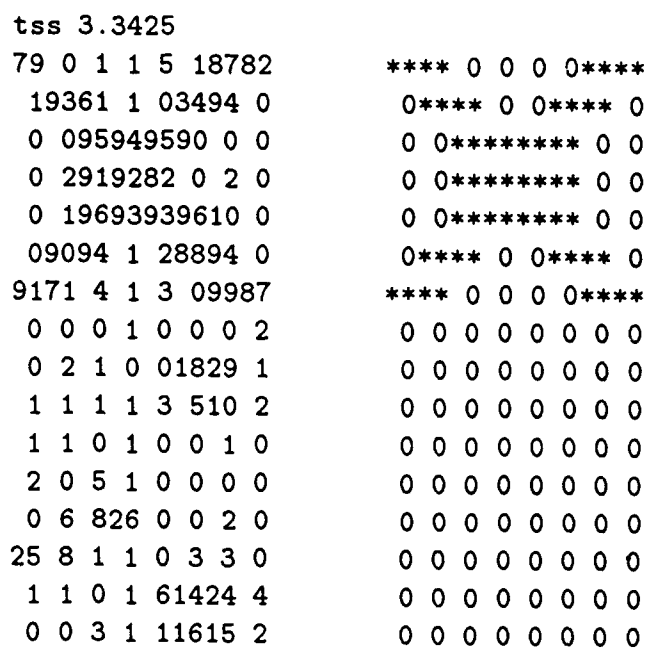


Figure 4.11: Output from level 220.20, pattern b1 (filtered).

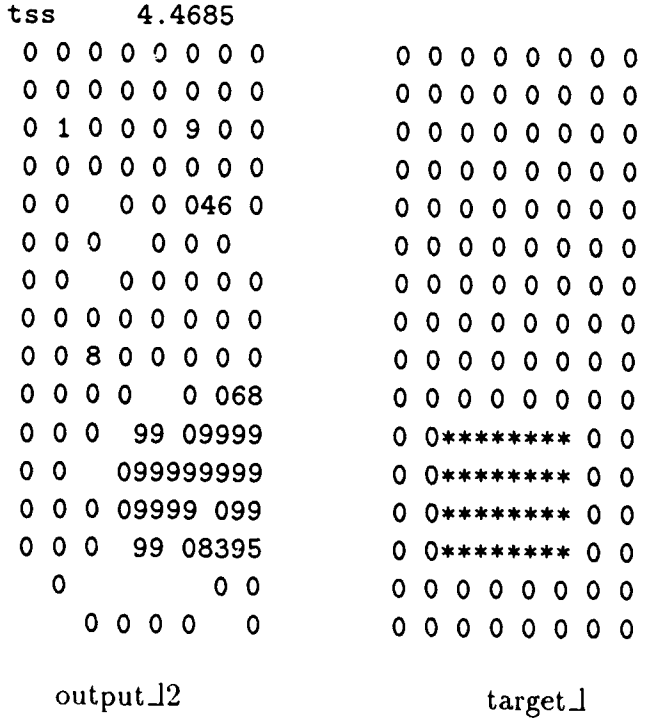


Figure 4.12: Output from level 200.20, pattern l2 (filtered).

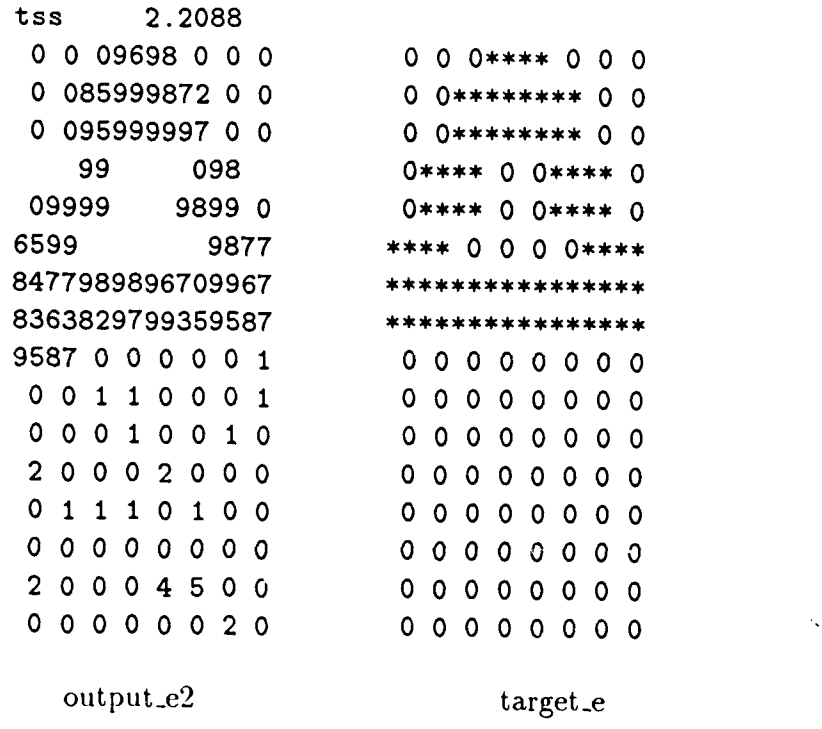


Figure 4.13: Output from level 200.20, pattern e2 (filtered).

level	200.00%	200.10%	200.15%	200.20%	
z1	0.0000	0.0227	24.686	18.224	
z2	0.0000	10.700	6.1002	4.3758	
a1	0.0003	0.0015	0.0671	7.3241	←
i1	5.0000	5.0123	11.044	12.694	
k1	1.0004	1.0008	1.0006	1.0672	
m1	14.906	14.913	14.884	15.095	
sum pss	20.91	31.65	57.78	58.78	
%increase					
over prev level	-	51	82	2	
total % incr	-	51	176	181	

← - outputs shown below (Fig. 4.19, page 47).

Figure 4.14: Pattern error for level 2 (no weights zeroed).

level	210.00%	210.10%	210.15%	210.20%	
z1	0.0000	0.0217	24.215	18.232	←
z2	0.0000	10.976	5.9540	4.1622	
a1	0.0003	0.0021	0.0785	7.2943	
i1	5.0001	5 0124	10.803	12.720	
k1	1.0004	1.0007	1.0006	1.0678	
m1	14.916	14.922	14.896	15.124	
sum pss	20.92	31.93	56.95	58.60	
%increase					
over prev level	-	53	78	3	
total incr	-	53	172	180	

← - outputs shown below (Fig. 4.20, pagereffig:pic2).

Figure 4.15: Pattern error for level 2 (weights $\leq$ +/- .10 zeroed).

the error improved by changing the bits. Selected patterns $z1, z2, a1, i1$, $k1, and m1$ show the range of errors. Notice that there is serious degradation on $a1$ when 6 additional bits were flipped (Fig. 4.14). Based on the theory of graceful degradation and the fact that this sharp change only happens sporadically, this would appear to be an unusual and highly sensitive 6-bit change.

This next one is even more unusual than the previous one. You expect to get more error when you distort the figure more (Fig. 4.15). You do not expect for the error to go back down when you change the inputs even farther from the proper pattern. Especially significant is that the two z patterns are very similar inputs (different bits are flipped), but they have grossly different output errors (Fig. 4.15, 4.16, 4.17).

The drastic changes after changing only a few bits leads us to believe that there is something peculiar about these nets. It is not shown in the figures, but most patterns followed the what you would consider a normal progression of error levels. Therefore, the percentage change in the summary line is of lesser consequence. PSS less than 1 is considered excellent. There is little trouble in making out the patterns with a pss of 1. After seeing the good results on distorted patterns, it would be a simple task for another net to polish up the outputs to make nearly perfect patterns.

From the *normal* patterns, those that did not make drastic changes, you start to see significant changes when 20% of the bits are wrong. This can not be taken as conclusive, since there are still those enormous errors in some patterns. Further study needs to determine which bits or combination of bits caused the high error, and how the net can be trained so as not to be so sensitive to specific

level	215.00%	215.10%	215.15%	215.20%	
z1	0.0001	0.0177	24.210	18.302	
z2	0.0001	10.437	5.9746	4.4909	
a1	0.0003	0.0030	0.0720	6.9355	
i1	5.0000	5.0134	10.787	12.701	←
k1	1.0004	1.0007	1.0005	1.0493	
m1	14.905	14.911	14.863	15.111	
sum pss	20.91	31.38	56.91	58.59	
%increase					
over prev level	-	50	81	3	
total incr	-	50	172	180	

← - Outputs shown below (Fig. 4.21, page 49).

Figure 4.16: Pattern error for level 2 (weights $\leq$ +/- .15 zeroed).

level	220.00%	220.10%	220.15%	220.20%	
z1	0.0000	0.0144	23.389	17.681	
z2	0.0000	8.8353	5.5288	4.3050	
a1	0.0004	0.0050	0.1031	8.2028	
i1	5.0000	5.0114	10.349	11.966	←
k1	1.0005	1.0008	1.0007	1.0502	
m1	14.891	14.897	14.846	15.061	
sum pss	20.9	29.76	55.21	58.27	
%increase					
over prev level	-	42	86	6	
total	-	42	164	179	

← - Outputs shown below (fig. 4.22, page 50).

Figure 4.17: Pattern error for level 2 (weights $\leq$ +/- .20 zeroed).

bits. It may help to train the net with barely distorted patterns. However, this will increase training time by $\mathcal{O}\ n$ (where n is the number of additional altered patterns used for each regular pattern).

The initial results for level 3 were too bad to even look at. Because of the high value for momentum the net did everything poorly. If you notice the graph of level 3 where momentum (Fig. 4.7) equals .9, you will see that severe oscillation controls the graph and it does not converge to anything useful. Notice that at momentum equals .3 the graph does converge and converges much better. The following results show the performance with that value for the momentum and training of only 60 epochs (Fig. 4.18), whereas all of the other levels were trained to 500.

Notice that before all of the pattern m's were around 14 and were the hardest patterns to learn. Here pattern m is less than 1 for most of the series, and still less than 2 at 20% distorted. Now, the blank shows the worst error of this series with the pattern $i1$ coming up the next worst. The tss for an entire 14 pattern test is less that 1.

The following series show the progression of output patterns as the inputs are distorted (Fig. 4.19, 4.20, 4.21, and 4.22). In the labels, the .00, .10, .15, and .20 are the percentages of bits that were flipped (i.e. 10%, 15%, 20%). There are several places where the changing certain bits caused drastic change in the pss. One (or more) of the bits in the distortion mask must trigger a sensitivity place in the net. Basically, you see graceful degradation, however there are also some that are not so graceful. These are all taken from level 2. All levels had these kind of progressions. These are used as an indicator of an overall problem, not something peculiar to level 2.

level	320.00%	320.10%	320.15%	320.20%
z1	0.0036	1.5857	1.2600	2.0154
z2	0.0036	0.9739	0.6046	0.2186
a1	0.0019	0.0186	0.9211	1.2559
i1	0.0024	0.0366	2.2879	0.7651
k1	0.0012	0.0071	0.0046	1.4771
m1	0.0050	0.0551	0.1387	1.6381
sum pss	.0177	2.677	5.217	7.370
%increase				
over prev level	-	167	94	41
total	-	167	293	637

Figure 4.18: Pattern error for level 3 (no weights zeroed).

pat_a1

200.00

tss .003

```
0 0 09999 0 0 0
0 099999999 0 0
099 0 0 0 099 0
9999 0 0 0 09999
9999 0 0 0 09999
099 0 0 0 099 0
0 099999999 0 0
0 0 09999 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

200.10

.0015

```
0 0 09999 1 0 0
0 099999999 0 0
099 0 0 0 099 0
9999 0 0 0 09999
9999 0 0 0 09999
099 0 0 0 099 1
0 099999999 0 0
0 0 09999 0 0 2
0 2 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

200.15

tss 0.0671

```
0 0 09999 1 0 0
0 098999999 0 0
093 0 0 0 081 0
9999 1 0 0 09999
9999 5 1 0 09999
094 010 3 099 6
0 099999995 0 0
0 0 09999 0 0 1
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 2 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

200.20

7.3241

```
0 0 19999 0 0 1
0 014999915 0 0
0 6 06831 020 0
9999532623 09999
9999303063589999
020 04554 076 0
0 128999914 5 0
0 0 29999 0 0 2
0 2 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

Figure 4.19: Pattern i1 progression, no weights zeroed.

pat_z2

210.00

```
tss 0.0000
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

210.10

```
10.9767
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 011 0 0
0 0 0 0 1 0 0 0
0 0 4 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 3 05070 0
0 0 0 1 0 29166
0 0 0 4 0 09386
0 075 015 29698
0 0 0299861 024
8641 0 0 8 0 1 0
6517 0 0 499 6 0
417 0 0 88925 0
```

```
210.15
tss    5.9540
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 1 0 0 0 3 0 0
0 0 0 0 0 0 0 0
0 0 2 0 0 0 1 0
0 0 1 0 0 0 1 0
0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 03162 0
0 0 0 0 0 37869
0 0 029 0 03522
0 096 0 1 06819
0 23044 237 0 2
3986 0 0 0 0 8 0
11 0 0 0 999 9 0
0 0 0 0 04821 0
```

```
200.20
4.1622
0 0 0 1 5 0 0 0
0 0 0 2 0 0 0 0
0 1 2 1 210 0 0
0 6 6 3 0 0 2 0
0 532 1 0 551 0
0 2 1 0 1 013 0
1 0 1 1 3 0 3 0
0 0 0 0 0 0 0 0
0 0 0 0 06116 0
0 0 0 020 32037
0 0 026 0 0 0 0
1 028 0 1 0 0 0
0 7 882 0 0 0 0
2621 021 0 0 2 0
6 0 0 046954654
0 0 0 0 0 866 0
```

Figure 4.20: Pattern z2 progression, weights $\leq +/-$ .1 zeroed.

pat_i1

215.00	215.10
tss 5.0000	5.0134
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 09999 0 0 0 0	0 09899 0 1 0 1
0 0 09999 0 099	0 0 19999 0 099
0 0 0 099 09999	0 0 0 199 09999
0 0 0 099999999	0 0 0 099999999
0 0 0 09999 099	0 0 0 09999 099
0 0 09999 0 099	0 1 09899 0 199
99 09999 0 0 0 0	99 29999 2 6 0 0
9999 0 0 0 0 0 0	9999 0 0 4 0 1 0

215.15	215.20
tss 10.7871	12.7011
13 0 0 0 0 6 0 0	11 0 0 0 0 0 0 0
0 0 0 0 036 0 0	0 0 0 0 0 0 0 0
0 0 6 0 496 0 0	0 0 0 0 0 0 0 0
0 0 1 0 4 0 0 0	0 0 0 0 0 0 0 0
0 09133 0 1 4 0	0 0 0 4 0 0 0 0
0 545 2 0 01834	0 063 0 0 0 0 0
7 038 0 0 0 144	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 09999 0 0 243	0 05487 09579 0
0 0449996 0 099	0 0 05199331099
0 0592599 09999	0 0 01099 09999
0 0 6 099829999	0 0 5 099419799
0 0 0 09999 099	0 0 2769984 099
017 29099 2 083	6 1 89996 0 057
994099985543 0 0	9929999867994071
9999 4 164 0 0 0	9999 5 2161796 1

Figure 4.21: Pattern i1 progression, weights $\leq$ +/- .15 zeroed.

pat_i1

220.00

tss 5.0000

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 09999 0 0 0 0
0 0 09999 0 099
0 0 0 099 09999
0 0 0 099999999
0 0 0 09999 099
0 0 09999 0 099
99 09999 0 0 0 0
9999 0 0 0 0 0 0
```

220.10

5.0114

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 09899 0 1 0 1
0 0 19999 0 099
0 0 0 199 09999
0 0 0 099999999
0 0 1 09999 099
0 1 09899 0 199
99 19999 3 4 0 0
9999 0 0 5 0 1 0
```

220.15

tss 10.3496

```
11 0 0 0 0 5 0 0
 0 0 0 0 040 0 0
 0 0 4 0 395 0 0
 0 0 2 0 3 0 0 0
 0 09137 0 0 4 0
 0 435 2 0 01835
 7 031 0 0 0 144
 0 0 0 0 0 0 0 0
 0 09999 0 0 139
 0 0449995 0 099
 0 0552099 09999
 0 0 5 099889999
 0 0 0 09999 099
 012 28999 1 081
994099985936 0 0
9999 3 160 0 0 0
```

220.20

11.9668

```
11 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 4 0 0 0 0
 0 054 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 05691 09370 0
 0 0 0549926 599
 0 0 0 899 09999
 0 0 3 099499699
 0 0 1709984 099
 4 1 99997 0 062
9923999871984269
9999 5 2181295 2
```

Figure 4.22: Pattern i1 progression, weights $\leq$ +/- .2 zeroed.

4.5 Change from zeroing low weights

There was no direct correlation between the value of the weights that were zeroed and a significant loss of pattern image. The net sensitivity is much more critical. It is imperative to know the pixels that will cause severe image failure before any of these nets can be useful. First, we see the number of weights that were zeroed (Fig. 4.23). These are the number of connections that are effectively eliminated. This is a significant reduction in time to evaluate an input. It does not however reduce the training time, because there is no indication of what to zero until the training is complete. The second figure shows the number of multiplications that each net normally runs during the training for one epoch of 14 patterns (Fig. 4.24). It will be discussed in the next section. It is shown here to illustrate the amount of training time, associated with each level. Remember that it may take from 100 to 500 cycles to insure that you have reached asymptotic limits of the training curve.

Fig. 4.25 shows the change when you only vary the zeroed values. There is very little change for any of the patterns, and you have reduced the floating point multiplications by 30%. Pattern $m1$ actually improved in some cases by zeroing the weights. Figure 4.24 shows the total number of floating point multiplications to test and to train (1 cycle for 1 pattern, all 14 patterns, and 500 epochs).

4.6 Is bigger better?

Finally, the answer to the question if bigger is better. Increasing the width improved the net more than increasing the depth. Even so, you are better off to

level 2 - 33,152 connections			
	weights zeroed	number zeroed	% changed
	.10	4447	13
	.15	6549	19
	.20	8613	25
level 3 - 49,664 connections			
	.10	9012	18
	.15	13541	27
	.20	17943	36

Figure 4.23: Weights zeroed in 2 levels and the percentage of total connections.

level	test	training(1 pat)	training (14 pats)	500 epochs
1	33152	82817	1.1m	579m
2	66304	165633	2.3m	1.2b
3	99456	248449	3.5m	1.7b
3out	132224	330369	4.6m	2.3b
4	132608	331265	4.6m	2.3b
4out	198144	405105	5.7m	2.8b
5	165760	414081	5.8m	2.9b

(m - million, b - billion)

Figure 4.24: Number of multiplications.

level	200.00%	210.00%	215.00%	220.00%
z1	0.0000	0.0000	0.0001	0.0000
z2	0.0000	0.0000	0.0001	0.0000
a1	0.0003	0.0003	0.0003	0.0004
i1	5.0000	5.0001	5.0000	5.0000
k1	1.0004	1.0004	1.0004	1.0005
m1	14.906	14.916	14.905	14.891
sum pss	20.91	21.01	20.91	20.9
%increase				
over prev level	–	.4	-.4	.4
total incr	–	.4	0	.4

Figure 4.25: Increasing the zeroes for level 2.

get another level deeper, because you use less cpu time than in adding a level than from widening (Fig. 4.24). Increasing the width of the net did improve the performance over the net that was deeper. With the time required to train the net, you could use a still deeper (higher level)net that would perform better. Look at Fig. 4.26, under columns 300.00 and 3out00.00. The wider net (3out00.00) did better than the deeper net (300.00) The wider net did better than the next level (400.00), but at greater cost. The floating point multiplications for "level 3out" were actually higher than for "level 4".

level	100.00	200.00	300.00	3out00.00	400.00	4out00.00
z1	0.0274	0.0000	0.0234	0.0036	0.0027	0.0013
z2	0.0274	0.0000	0.0234	0.0036	0.0027	0.0013
a1	8.0000	0.0003	0.0337	0.0019	0.0034	0.0010
i1	2.0005	5.0000	0.0688	0.0024	0.0070	0.0018
k1	10.000	1.0004	0.0322	0.0012	0.0035	0.0008
m1	46.000	14.906	0.1353	0.0050	0.0104	0.0025
sum pss	66.06	20.97	.3168	.0177	.0297	.0087
% reduction over prev level	-	68	98	94	-69	71

Figure 4.26: Progression for levels 1-4.

Chapter 5

Summary and Future Work

This chapter is made up of two main section. The first will provide the overall summary of the results. This will give you the short synopsis of the results from chapter 4. Things that are of a follow-on nature from this study for the near future, are located with the summary. The second section is the ' what's next ' part. It will, also, give a brief look to the more distant future of things that will be needed in the long term.

5.1 Summary

Momentum - One of the most critical elements in reducing total training error and the effectiveness of the entire system. With momentum = .3, total error for an entire group was less than the largest single pattern error. There needs to be found a value, set of values, or rule of thumb to describe the optimum momentum for the entire series of nets.

Seed – The second critical value is the seed. One seed was found to be the cause of serious deviation from the training curve of the other 5 runs. Final error was 4 times that of the rest of the runs. The other runs were extremely tight in the graph of the training error. This was run before momentum was found to be so critical and may well be a mute point after that is solved.

Sensitivity – A few patterns were grossly distorted after a small change in some parameter or flipping some bits. There is an unusual sensitivity that does not allow for graceful degradation for all patterns. These patterns decayed gracefully up to a point, but then something, as yet unexplained, occurred. Training with distorted patterns may help overcome this problem. This indicates that a few bits may control a portion of the net, and disallow graceful degradation. Training that uses mildly distorted patterns may help fix this problem.

Training cycles – More patterns proportionately increase the error during the first few training cycles. Larger size groups have surpassed smaller in total error in several places in the training curves. Most of the training occured in the first 10 epochs, with little additional improvement after 100 epochs. Since training error can increase after more training, a heuristic is needed to determine when to stop the training.

Pattern distortion – Outputs showed gradual degradation through the 20went from less than 1 (tss) to more than 20 over one increase in distortion. There were other patterns that would improve after distortion was increased even more. This leads me to believe that the nets were highly sensitive to specific bits. Training with mildly distorted patterns may also improve performance here.

Weight adjustment – Similar results as were noticed with pattern distortion, including the gross changes in tss. Other than the patterns in question for net sensitivity, 30% of the weights could be zeroed out with little loss of performance. Weights were zeroed where the absolute value was less than or equal .20 .

Time – Time studies showed that it takes one hour of cpu time to train a level 3 net o ı a 14-pattern group. The nets used were intended to be simple and flexible. Therefore, little should be done initially to improve the times until such time that the other problems are resolved.

5.2 Future Work

The most significant improvement for this area is the development of a parallel implementation for the nets. Parallel implementations have been used, but are not readily available. That would speed up the research process immensely. You could almost see the net learning. Problems could be addressed more quickly and solutions initiated. By the time, you have waited hours to get a little bit of data, you have lost the train of thought that you intended to explore. At the present, you can do little more than get it started and let it run all day.

One of the problems, of a study of this depth, is that the training cycles take so long. For a significant statistical analysis, you need many more runs of each size group and on each net. A small sample gives you a feel for what you can expect, but it is not statistically valid. Each of these tests should be run for at least 100 training runs, if not 1000. Each run should be for 1000 cycles, to insure there was not a late drop. This would give you enough of a sample to make a solid analysis. Be aware that the memory requirements to save the weights for

the other testing would be enormous.

These tests showed by force what the problems and faults were for the nets. Mathematical proofs are needed to give final and complete results. It is but a first step to note the behavior of the net, now we need to show more than speculation as to the causes.

Heuristics methods for determining the seed for the initial random weights or for knowing when to stop training and change seeds are needed. The time required to fully train a net is wasted if the net performs poorly. Such a method would allow the user to cut his losses earlier.

BIBLIOGRAPHY

Bibliography

[1] Abu-Mostafa, Y. S., and Psaltis, D. Optical Neural Computers, *Scientific American*, Mar 1987, pp 88-95

[2] Chua, L.O. and Yang, L. Cellular Neural Networks: Theory, *IEEE Transactions on Circuits and Systems*, May 1986 pp. 1257-1275

[3] Chua, L.O. and Yang, L., Cellular Neural Networks: Application, *IEEE Transactions on Circuits and Systems*, May 1986 pp. 1275-1285

[4] Fahlman, S. E., and Hinton, G. E., Connctionist Architecture for Artificial Intelligence, *Computer*, vol 21, Jan 1987, pp 100-109

[5] Feldman, J. A., Fanty, M. A., and Goddard, N. H., Computing with Neural Networks, *Computer*, vol 21, Mar 1988, pp 91-102

[6] Fukushima, K., A Neural Network for Visual Pattern Recognition, *Computer*, vol 21, Mar 1988, pp 65-70

[7] Graf, H., Jackel, L., and Hubbard, W.,VLSI Implementation of Neural Network Model,*Computer*, vol 21, Mar 1988, pp 41-63

[8] Grossberg, S. ed., *Neural Networks and Natural Intelligence*, MIT Press, Cambridge, Mass., 1988

[9] Gorman, C., Putting Brainpower in a Box, *Time*, vol 132, Aug 1988

[10] Kohonen, T. *Associative Memory A System-Theoretical Approach* Springer-Verlag, New York, NY, 1977

[11] Kohonen, T. The "Neural" Phonetic Typewriter, *Computer*,vol 21, Mar 1988, pp 11-22

[12] Kuperstein, M., Neural Model of Adaptive Hand-eye Corrdination for Single Postures, *Science*, vol 239, Mar 1988, pp 1308-1311

[13] MacLennan, B. J., Technology-Independent Design of Neurocomputers: The Universal Field Computer, published in the Proceedings of the IEEE First Annual International Conference on Neural Networks, Jun 1987

[14] Minsky, M. *Perceptrons* MIT Press, Cambridge, Mass., 1987

[15] Rumelhart, D. E. and Zipser, *Competitive Learning Models*, 1985, pp 86-87, in Grossberg, S., *Competitive Learning: From interactive activation to adaptive resonance*, 1987, in *Neural Networks and Natural Intelligence* edited by S. Grossberg, 1988

[16] Rumelhart, D. E., McClelland, J. L., and PDP Research Group, *Explorations in Parallel Distributed Processing*, MIT Press, Cambridge, Mass., 1988

[17] Rumelhart, D. E., McClelland, J. L., and PDP Research Group *Parallel Distributed Processing - Explorations in the Microstructure of Cognition*, MIT Press, Cambridge, Mass., 1988

[18] Shriver, B. D., Artificial Neural Systems, *Computer*, vol 21, Mar 1988, pp 8-9

[19] Smith, M. panelist in the seminar "Case Studies in the Commercialization of Neural Networks," at IEEE Int'l Conf on Neural Networks, San Diego, CA, 1988

[20] Tank, D. W., Collective Computation in Neuron-like Circuits, *Scietific American*, vol 257, Dec 1987, pp 104-115

[21] Widrow, B., and Winter, R., Neural Nets for Adaptive Filtering, *Computer*, vol 21, Mar 1988, pp 25-39

APPENDIX

Patterns and Targets

=-=

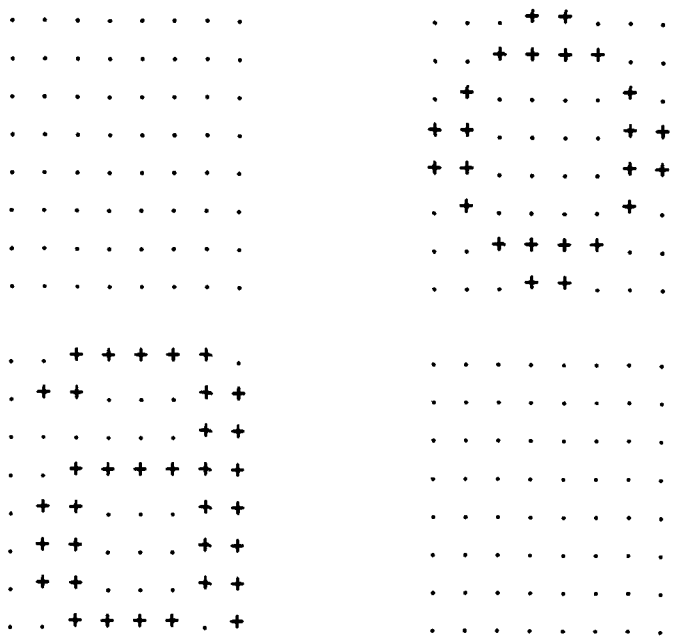
Pattern   Target

pat_z

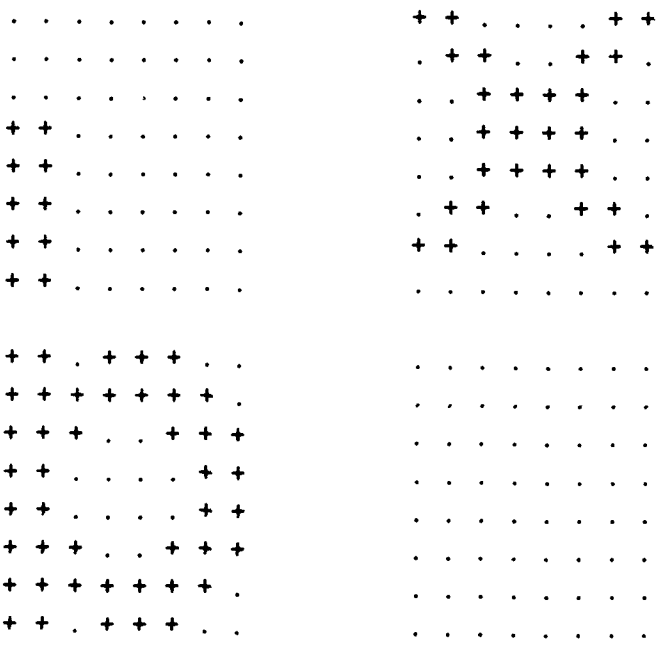
```
. . . . . . . .     . . . . . . . .
. . . . . . . .     . . . . . . . .
. . . . . . . .     . . . . . . . .
. . . . . . . .     . . . . . . . .
. . . . . . . .     . . . . . . . .
. . . . . . . .     . . . . . . . .
. . . . . . . .     . . . . . . . .
. . . . . . . .     . . . . . . . .

. . . . . . . .     . . . . . . . .
. . . . . . . .     . . . . . . . .
. . . . . . . .     . . . . . . . .
. . . . . . . .     . . . . . . . .
. . . . . . . .     . . . . . . . .
. . . . . . . .     . . . . . . . .
. . . . . . . .     . . . . . . . .
. . . . . . . .     . . . . . . . .
```

‘‘ . ’’ \ \ --\ \ input=0   ‘‘ + ’’ \ \ --\ \ input=1

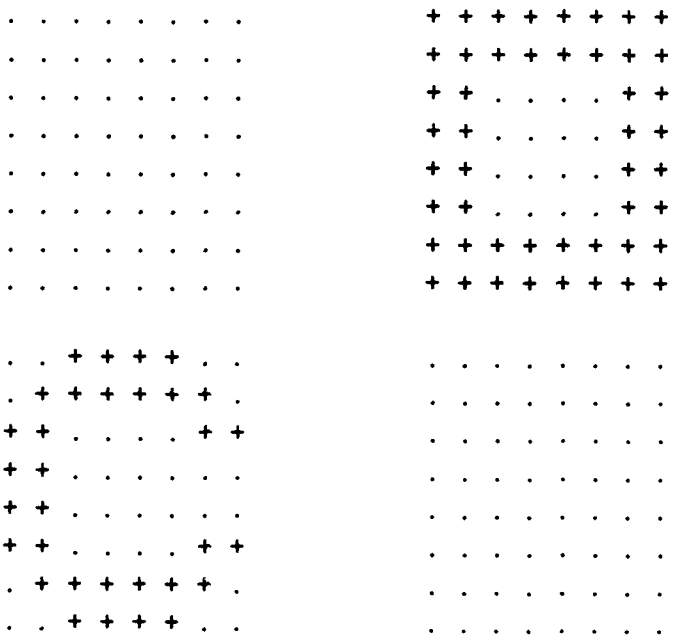
pat_a



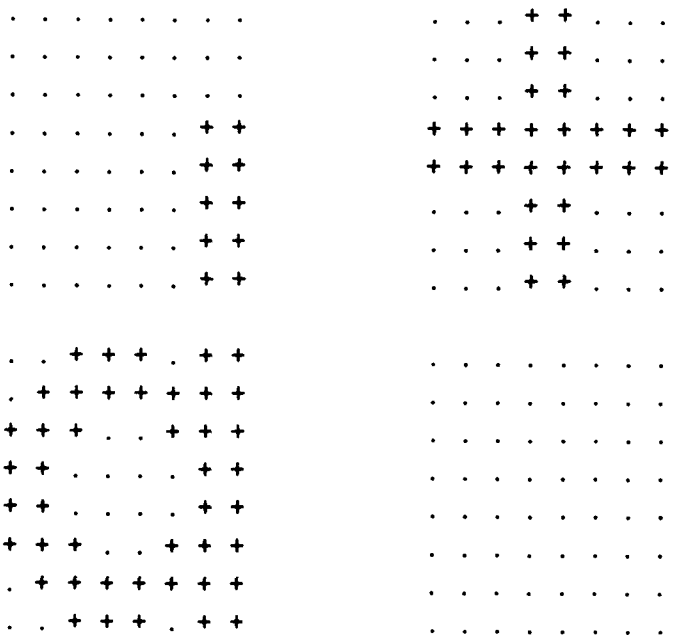
pat_b



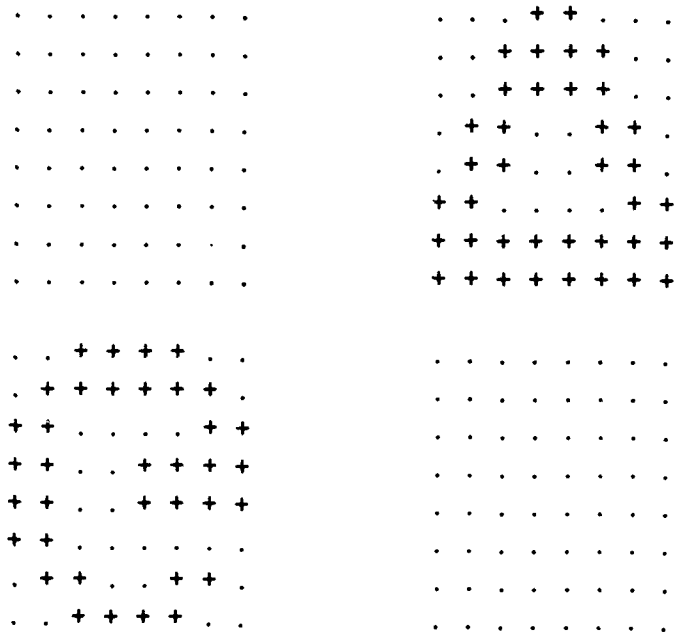
pat_c



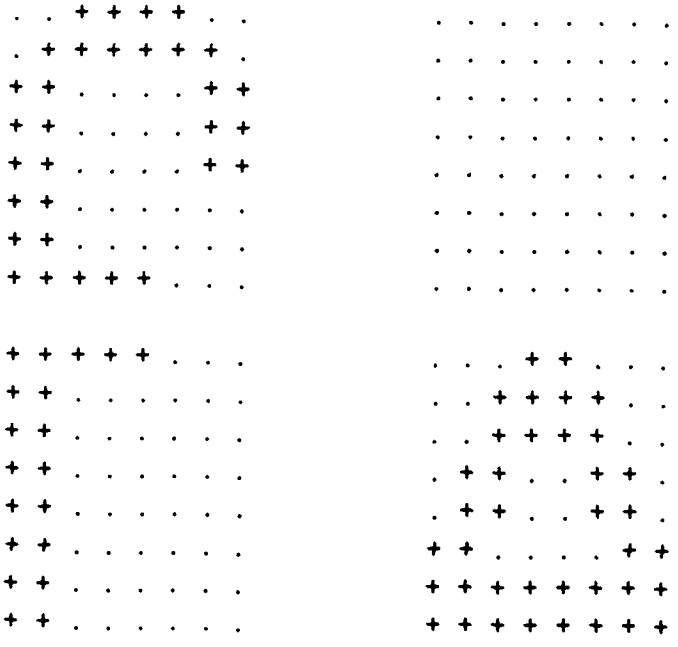
pat_d



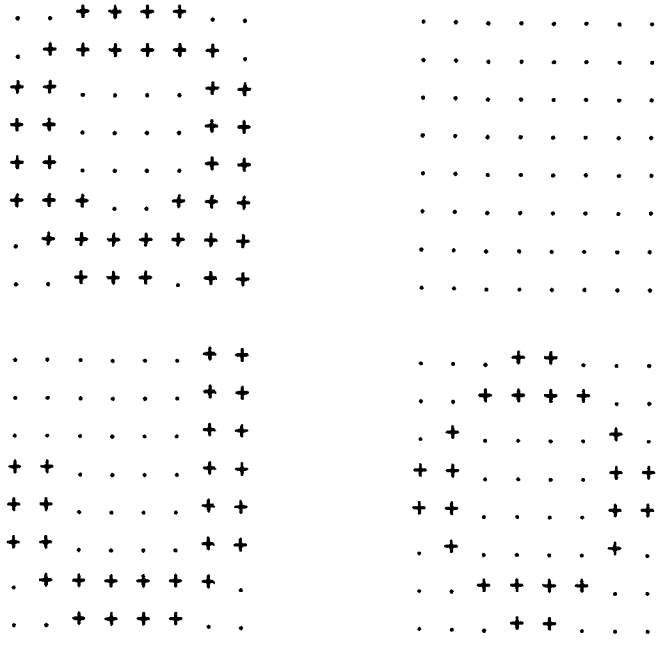
pat_e



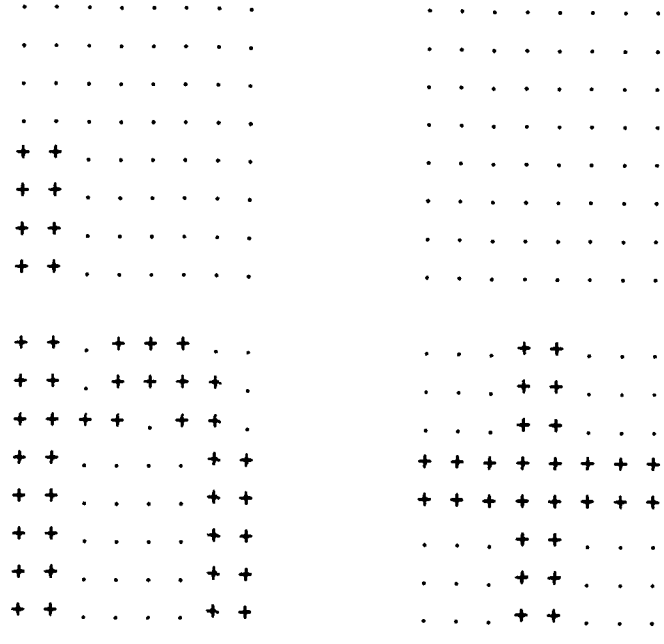
pat_f



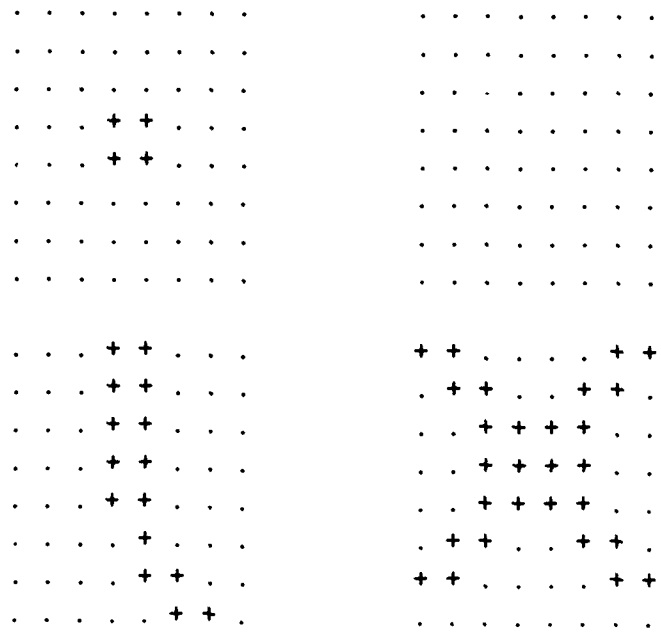
pat_g



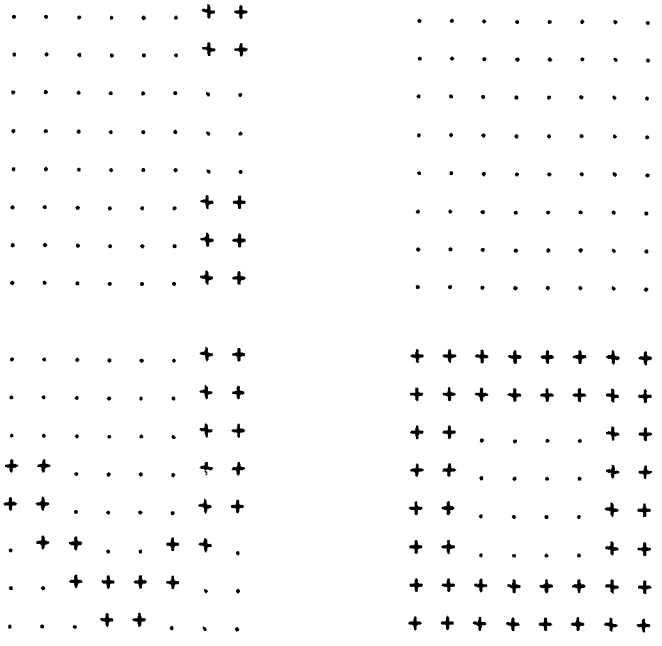
pat_h



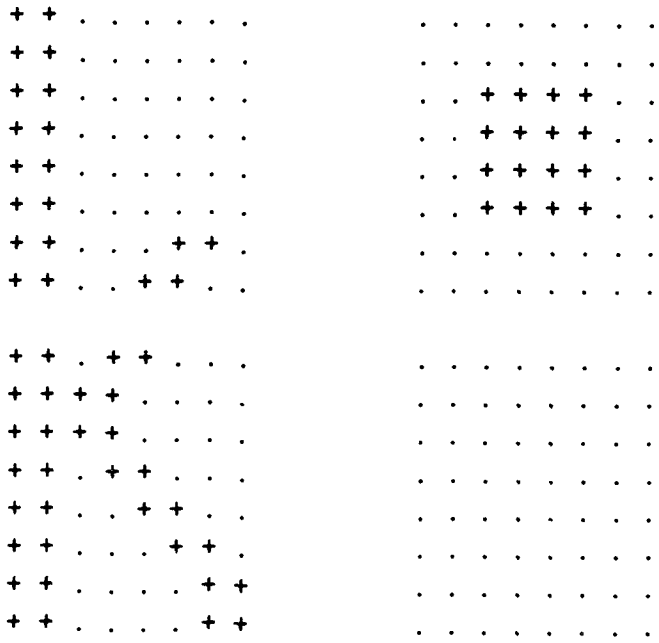
pat_i



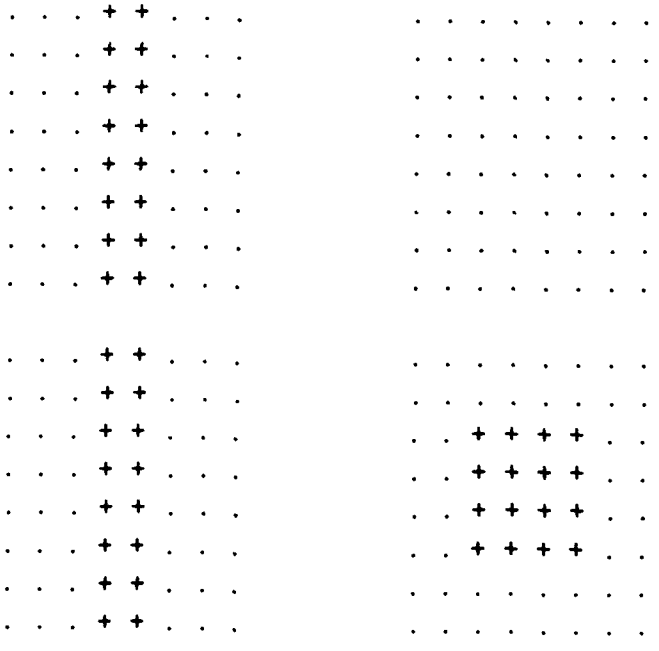
pat_j



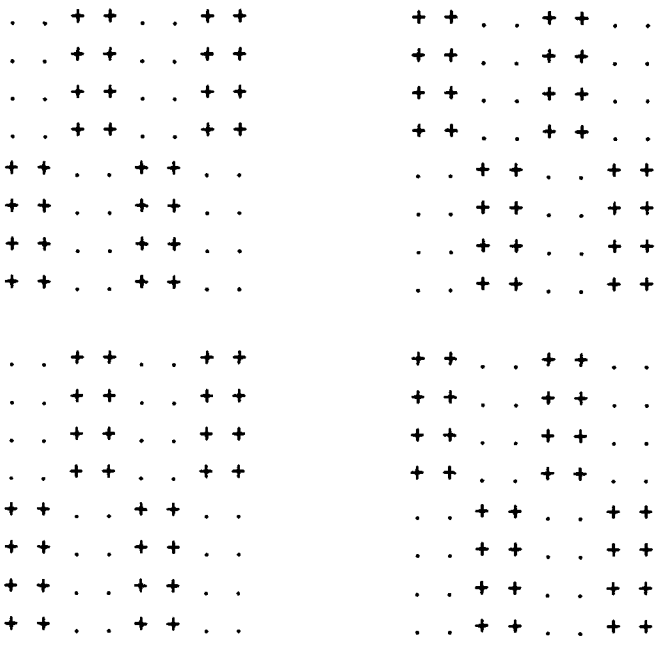
pat_k

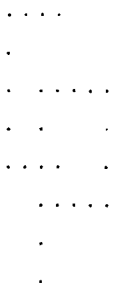


pat_l



pat_m





VITA

Vita

Clarence W. Potter, Sr. [redacted] He attended elementary school at Reagan Elementary Schoo[redacted] He graduated from [redacted] High School [redacted] May of 1974. His Bachelor of Business Administration (Accounting) Degree was awarded in August of 1978 from the University of Texas at Arlingtion.

He was commissioned a second lieutenant in the United States Army in September of 1978. Over the last 10 years, he has been stationed in Texas, Korea, Georgia, and North Carolina. While commanding an Air Defense Artillery Battery, with the 82nd Airborne Division, he was selected for participation in the fully funded graduate school program. He was accepted into the Master's program in computer science at the University of Tennessee.

He married [redacted] God blessed them with a son, [redacted]

After graduation, Captain Potter [redacted] His duties there will include finding ways that Artificial Intelligence applications will assist in improving the data systems input to the decision making process within the Army logistics system.